НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»

Факультет Інформатики та обчислювальної техніки

Кафедра обчислювальної техніки

До захисту допущено:
Завідувач кафедри
_____ Сергій СТІРЕНКО
«___»_____2020 р.

Дипломний проєкт

на здобуття ступеня бакалавра

за освітньо-професійною програмою «Комп'ютерні системи та мережі»

спеціальності 123 «Комп'ютерна інженерія»

на тему: «Модуль моніторингу стану розподіленої інформаційної системи»

Виконав:
студент IV курсу, групи ІО-61
Лисенко Дмитро Вадимович _____

Керівник:
Доцент, кандидат технічних наук,
Болдак Андрій Олександрович _____

Консультант з нормоконтролю:
Професор, доктор технічних наук,
Сімоненко Валерій Павлович _____

Рецензент:
Доцент, кандидат наук,
Ліщук Катерина Ігорівна _____

Засвідчую, що у цьому дипломному проєкті немає запозичень з праць інших авторів без відповідних посилань.
Студент (-ка) _____

Київ – 2020 року

Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»
Факультет Інформатики та обчислювальної техніки
Кафедра обчислювальної техніки

Рівень вищої освіти – перший (бакалаврський)

Спеціальність – 123 «Комп'ютерна інженерія»

Освітньо-професійна програма «Комп'ютерні системи та мережі»

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ Сергій СТІРЕНКО
«___»_____2020 р.

ЗАВДАННЯ

на дипломний проєкт студенту

Лисенко Дмитро Вадимович

1. Тема проєкту «**Модуль моніторингу стану розподіленої інформаційної системи**», керівник проєкту Болдак Андрій Олександрович, доцент, кандидат технічних наук, затверджені наказом по університету від «07» травня 2020 р. №1081-С

2. Термін подання студентом проєкту _____

3. Вихідні дані до проєкту: технічна документація, теоретичні та статистичні дані

4. Зміст пояснювальної записки: Опис предметної області, дослідження методики автоматизації налаштування системи моніторингу, програма модуля моніторингу

5. Перелік графічного матеріалу (із зазначенням обов’язкових креслеників, плакатів, презентацій тощо)

6. Консультанти розділів проєкту

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Нормоконтроль	Сімоненко В.П., професор, доктор технічних наук		

7. Дата видачі завдання _____

Календарний план

№ з/п	Назва етапів виконання дипломного проєкту	Термін виконання етапів проєкту	Примітка
1.	*Затвердження теми роботи*	*01.09.2019*	
2.	*Вивчення та аналіз завдання*	*01.09.2019-31.12.2019*	
3.	*Розробка архітектури та загальної структури системи*	*01.01. 2020-29.02. 2020*	
4.	*Програмна реалізація системи*	*01.03. 2020-31.03. 2020*	
5.	*Оформлення пояснювальної записки*	*01.04. 2020-24.05. 2020*	
6.	*Захист програмного продукту*	*25.04. 2020*	
7.	*Передзахист*	*26.05. 2020*	
8.	*Захист*	*18.06. 2020*	

Студент Дмитро ЛИСЕНКО

Керівник Андрій БОЛДАК

Анотація

В бакалаврській дипломній роботі реалізовано модуль для моніторингу стану розподіленої інформаційної системи, призначений для збору та відображення даних про стан вузлів розподіленої інформаційної системи.

Модуль виконує настроювання допоміжних компонентів на серверах, які відправляють дані на один комп'ютер, де відбувається їх обробка та відображення в зручному для людини вигляді. Програмний продукт створений на платформі Node.js з використанням стеку ELK. Працювати з модулем можна через консоль або графічний інтерфейс.

Для тестування модуля моделювалась розподілена система у програмному забезпеченні Docker.

Annotation

In this work for a Bachelor's Degree, the module for monitoring the state of the distributed information system is implemented. It is intended for collecting and displaying data about the state of the nodes of the distributed information system.

The module configures additional components on servers that send data to one computer, where the data are processed and displayed in a human-friendly way. The software is created with the Node.js engine and the ELK stack. You can work with the module through a console or graphical interface.

To test the module, a distributed system was simulated in the Docker software.

№ рядка	Формат	Позначення	Найменування	Кільк.	Примітка
1			Документація загальна		
2			Розроблена заново		
3					
4	А4	ІАЛЦ.466514.001 ОА	Модуль моніторингу стану	1	
5			розподіленої інформаційної системи		
6			Опис альбому		
7					
8	А4	ІАЛЦ.466514.002 ТЗ	Модуль моніторингу стану	3	
9			розподіленої інформаційної системи		
10			Технічне завдання		
11					
12	А4	ІАЛЦ.466514.003 ПЗ	Модуль моніторингу стану	60	
13			розподіленої інформаційної системи		
14			Пояснювальна записка		
15					
16	А4	ІАЛЦ.466514.004 Д1	Модуль моніторингу стану	1	
17			розподіленої інформаційної системи		
18			Алгоритм роботи модуля		
19					
20	А4	ІАЛЦ.466514.005 Д2	Модуль моніторингу стану	1	
21			розподіленої інформаційної системи		
22			Діаграма класів модуля		
23					
24	А4	ІАЛЦ.466514.006 Д3	Модуль моніторингу стану	1	
25			розподіленої інформаційної системи		
26			Діаграма сутностей		
27					
28					
29					
30					
31					
32					

					ІАЛЦ.466514.001 ОА		
Зм.	Арк.	№ докум.	Підпис	Дата			
Розробив		Лисенко Д. В.			Модуль моніторингу стану розподіленої інформаційної системи Опис альбому	Літ.	Аркуш 1 / Аркушів 1
Перевірив		Болдак А. О.					
Реценз.						НТУУ КПІ ім. Ігоря Сікорського, ФІОТ, ІО-61	
Н. контр.		Сімоненко В. П.					
Затвердив							

Технічне завдання до *дипломної* роботи

ЗМІСТ

1. НАЙМЕНУВАННЯ ТА ОБЛАСТЬ ЗАСТОСУВАННЯ

Дане технічне завдання розповсюджується на розробку модуля моніторингу стану розподіленої інформаційної системи. Даний модуль інтегрується у вже існуючі розподілені системи.

2. ПІДСТАВИ ДЛЯ РОЗРОБКИ

Підставою для розробки є завдання на виконання роботи кваліфікаційно-освітнього рівня «бакалавр комп'ютерної інженерії», затверджене кафедрою спеціалізованих комп'ютерних систем Національного технічного Університету України «Київський Політехнічний інститут ім. Ігоря Сікорського».

3. МЕТА ТА ПРИЗНАЧЕННЯ РОЗРОБКИ

Метою даного проекту є розробка модуля для моніторингу стану розподіленої інформаційної системи.

4. ДЖЕРЕЛА РОЗРОБКИ

Джерелами розробки є науково-технічна література по розподіленим системам і обробці даних, публікації в періодичних виданнях, публікації в Інтернеті по даним питанням.

5. ТЕХНІЧНІ ВИМОГИ

5.1. Вимоги до розроблюваного продукту

- Інтеграція – модуль повинен встановлюватися у вже існуючій розподіленій інформаційній системі.
- Автоматизація інтегрування – інтегрування відбувається за участі людини, проте його більша частина виконується автоматично.

- Командний та графічний інтерфейси – оператор обирає один з даних двох інтерфейсів роботи з модулем.

5.2. Вимоги до вузлів розподіленої системи

5.2.1. Вимоги до програмного забезпечення вузлів

- Операційна система Windows або Linux.
- Telnet сервер.

5.2.2. Вимоги до апаратного забезпечення вузлів

- Комп'ютер на базі процесора Intel Cour i3 і вище.
- Оперативної пам'яті не менше 1 Гбайт.
- Вільне місце на жорсткому диску не менше 800 Мбайт.
- Підключення до Інтернету.

5.3. Вимоги до центрального сервера

5.3.1. Вимоги до програмного забезпечення сервера

- Операційна система Windows або Linux.
- Платформа Node.js 12.0.0 і вище.
- JDK 1.8.0 і вище.
- Стек ELK – Elasticsearch, Logstash, Kibana.

5.3.2. Вимоги до апаратного забезпечення сервера

- Комп'ютер на базі процесора Intel Cour i3 і вище.
- Оперативної пам'яті не менше 4 Гбайт.
- Вільне місце на жорсткому диску залежить від кількості даних, які збираються.
- Підключення до Інтернету.
- Підключення до серверів через локальну мережу або Інтернет.

ЗМІСТ

ВСТУП

На сьогоднішній день розповсюджена концепція розподілених систем. Необхідність таких систем виникає внаслідок великого обсягу задач, з якими один сервер не може впоратись. Прикладами таких систем можуть бути розподілені системи для паралельного обчислення, паралельної обробки зовнішніх запитів від клієнтів у клієнт-серверній організації обміну даними, виокремлення серверів під окремі задачі, тощо.

В невеликих системах можна обійтись без моніторингу, проте при великій кількості серверів стає важче слідкувати за працездатністю системи, адже вихід з ладу або неправильна робота одного із серверів можуть не відразу стати явними. Таким чином виникає необхідність моніторингу такої системи.

Наразі існують різні інструменти для даної задачі, популярним прикладом є стек ELK (Elasticsearch, Logstash, Kibana). Даний стек збирає дані з усіх серверів на одному комп'ютері та відображає їх в зручному для людини вигляді. Але для його роботи необхідно виконати настройку відповідних компонентів на кожному сервері, що може бути нудним та довгим заняттям при великій кількості серверів. Та сама проблема виникає при настройці інших інструментів для моніторингу.

В даній роботі розроблюється модуль, який дозволяє виконати настройку стеку ELK в автоматизованому режимі. Настройка відбувається з одного центрального комп'ютера на всіх серверах через мережу. Оператору необхідно лише переконатися у доступі до всіх серверів через мережу та під час налаштування вводити додаткові параметри, які модуль буде запитувати.

РОЗДІЛ 1.
НАЛАШТУВАННЯ МОНІТОРИНГУ РОЗПОДІЛЕНОЇ СИСТЕМИ

1.1. Розподіленні системи та необхідність моніторингу

Розподілені системи дуже розповсюдженні в сучасному світі. Можна навести наступні приклади використання розподілених систем:

- розпаралелювання однієї задачі на декількох серверах;
- розподіл великої кількості однорідних запитів серед декількох серверів;
- виокремлення серверів під окремі задачі (DNS та DHCP сервера, почтовий сервер, FTP сервер, тощо).

При відсутності моніторингу єдиною ознакою якихось проблем буде звернення від користувача з відповідною проблемою. Багато користувачів при неправильній роботі системи не будуть звертатися до технічної підтримки, а замість цього можуть почати користуватися іншою системою, що може призвести до втрати клієнтів. Коли ж користувач звернеться з якоюсь проблемою, при відсутності зібраних даних моніторингу буде важко знайти першопричину проблеми.

Отже, за такими системами необхідно слідкувати, щоб при виході з ладу або некоректній роботі одного із серверів, була швидка реакція від адміністратора відносно усунення виниклої проблеми.

Для моніторингу працездатності достатньо налаштувати сервери на відправку сигналу час від часу на сервер, який відповідає за моніторинг всієї системи (для спрощення в подальшому будемо називати такий сервер "центральний сервер"). При відсутності сигналу, центральний сервер буде повідомляти адміністратора (наприклад, по електронній пошті) по електронній пошті.

Для моніторингу коректності роботи серверів необхідно налаштувати більш складну систему. Для цього необхідно збирати інформацію про завантаженість ресурсів серверів, затримку відповідей на запити, кількість помилкових результатів (наприклад кількість відповідей з кодами 400 для HTTP серверу), логи роботи різних програм, тощо. Крім збору такої інформації її необхідно аналізувати та повідомляти адміністратора тільки в необхідних випадках[1] (наприклад, багато помилкових результатів або ресурси перевантаженні).

1.2. Існуючі рішення моніторингу

Наразі існує багато рішень для моніторингу таких систем[2]. В даному підрозділі розглянемо деякі з них, а в наступному розглянемо загальну проблему автоматизації налаштування моніторингу.

Стек ELK. Назва стеку – це перші літери трьох компонентів, які входять до цього стеку[3].

Компоненти стеку:

- Elasticsearch – платформа для зберігання інформації у форматі JSON та подальшому її пошуку. Встановлюється на центральному сервері;
- Logstash – платформа для збору інформації з різних джерел. У даному стеку вона пересилає цю інформацію в Elasticsearch. Якщо біти (англ. beats) не використовуються, то встановлюються на серверах, з яких збирається інформація;
- Kibana – платформа для візуалізації даних, які зберігаються в Elasticsearch. Встановлюється на центральному сервері;
- Біти (англ. beats) – збирають інформацію подібно до Logstash, проте вони мають менше можливостей і за рахунок цього використовують менше ресурсів. Цей різновид компонентів з'явився після формування стеку, тому вони не входять до його назви. Можуть використовуватись замість Logstash, або в

комбінації з ним. В першому випадку вони відправляють дані напряму до Elasticsearch, в другому через Logstash для приведення даних у потрібний вигляд. В цьому випадку біти встановлюються на вузли розподіленої системи, а Logstash на центральному сервері для зменшення використання ресурсів вузлів розподіленої системи.

На рис. 1.1 зображено приклад послідовності передачі даних моніторингу у системі з ELK стеком, а на рис. 1.2 приклад візуалізованих даних моніторингу у Kibana.

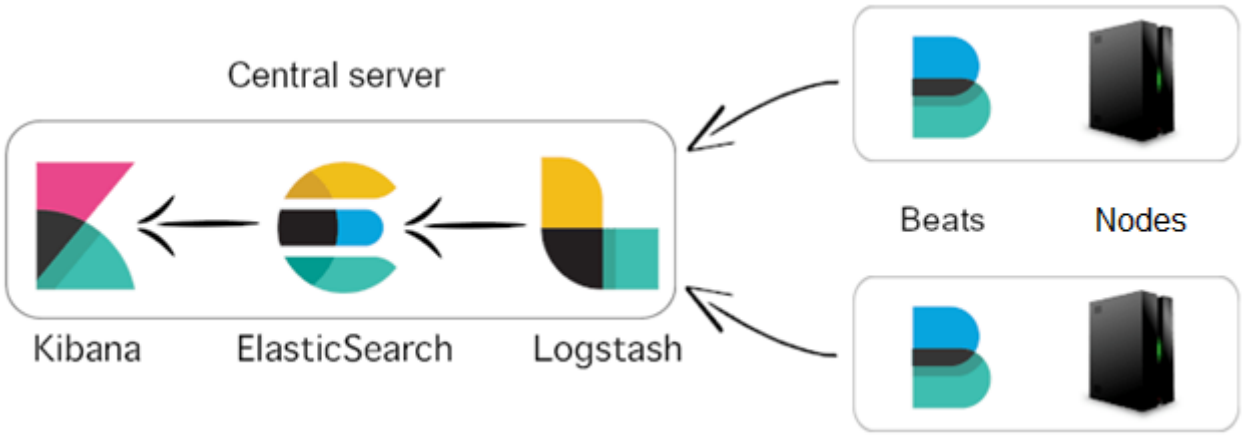


Рис. 1.1. Робота стеку ELK з використанням Logstash та бітів (англ. beats) в розподіленій системі

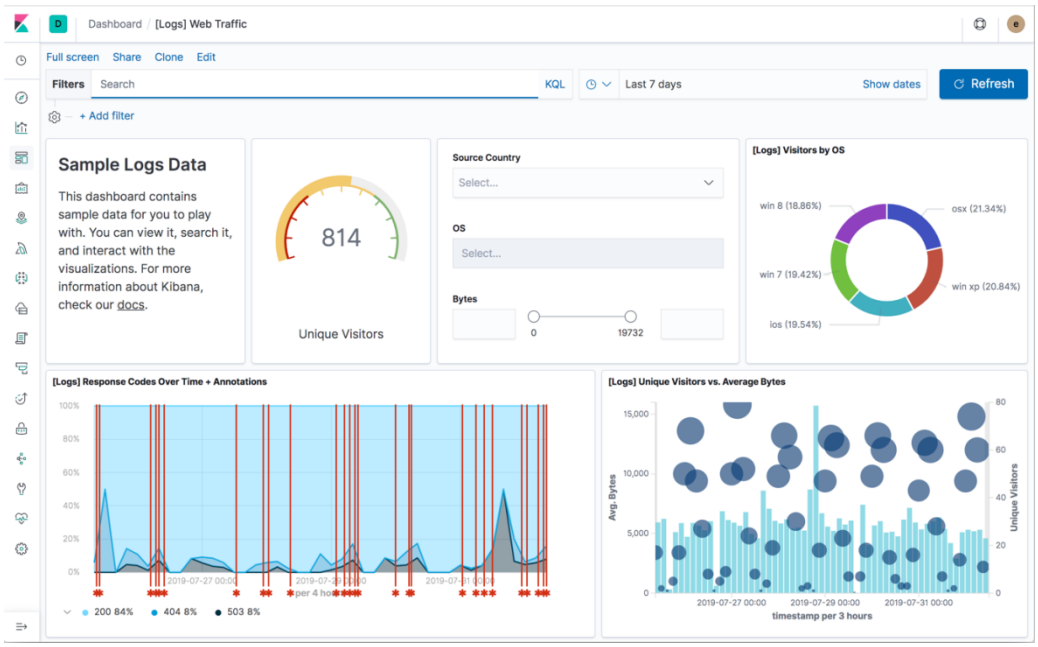


Рис. 1.2. Приклад візуалізованих даних моніторингу у Kibana

Даний стек дозволяє виконувати моніторинг розподіленої системи, в якій сервери можуть мати різне програмне забезпечення. Так для FTP серверу можна серед іншого слідкувати за використанням постійної пам'яті, а для серверу хмарних обчислень за завантаженістю процесорів. Також різні програмні засоби мають різні логи, які також можуть бути специфічні для кожного серверу. Також різні сервери можуть мати різні операційні системи, які можуть підтримуватись одночасно даним стеком.

Для використання цього стеку необхідно на центральному сервері встановити Elasticsearch, Kibana і можливо Logstash, налаштувати їх у відповідних файлах конфігурації та запустити на виконання. На всіх інших серверах необхідно зробити те саме для бітів (англ. beats) або Logstash. При великій кількості серверів цей процес може зайняти багато часу оператора.

Zabbix[4]**.** Дане рішення має два основні способи налаштування:

- інсталювання операційної системи з встановленими компонентами Zabbix. Для цього необхідно завантажити відповідний CD образ Zabbix та встановити його на кожному сервері. При використанні даного способу встановлюється операційна система CentOS 8, компоненти Zabbix та відкриваються відповідні порти для подальшого налаштування з центрального сервера;
- інтеграція у вже існуючу систему. При даному способі необхідно завантажити на налаштувати відповідні агенти на кожному сервері. Дані агенти використовують мало ресурсів та відправляють інформацію про стан сервера на центральний сервер.

Використання першого способу не підходить, коли необхідно використовувати операційну систему відмінну від CentOS 8 та/або коли сервер вже налаштований на його основні задачі і повторне налаштування може зайняти багато часу оператора та бути недоступний під час

встановлення нової операційної системи та повторного налаштування сервера під основні задачі. Також налаштовувати моніторинг конкретних метрик серверів все ще потрібно.

При використанні другого способу знову виникає проблема налаштування кожного сервера окремо, що може зайняти багато часу оператора.

На рис. 1.3 зображено приклад налаштованого Zabbix.

Рис. 1.3. Приклад налаштованого Zabbix

Nagios[5] складається з двох основних компонентів:

- Nagios Core (безкоштовний варіант) / Nagios XI (комерційний з більшою кількістю можливостей) – програмне забезпечення для центрального серверу. Забезпечує зберігання та візуальне відображення інформації про розподілену систему;
- Агенти – дані компоненти встановлюються та налаштовуються на розподілених серверах і використовують невелику кількість ресурсів. Вони збирають те передають інформацію до центрального серверу.

Як і в попередніх рішеннях, у Nagios також існує питання конфігурації кожного серверу окремо, оскільки необхідно налаштувати агентів на

кожному з них. Проте в даному випадку є засоби для легкого зберігання налаштування для повторного використання або для відновлення у випадку втрати даних на сервері. Це значно спрощує налаштування однорідної розподіленої системи, проте для неоднорідних систем (тобто агентів на кожному з них необхідно налаштовувати по різному) ця процедура також займає ресурси оператора.

Приклад налаштованого Nagios зображено на рис. 1.4.

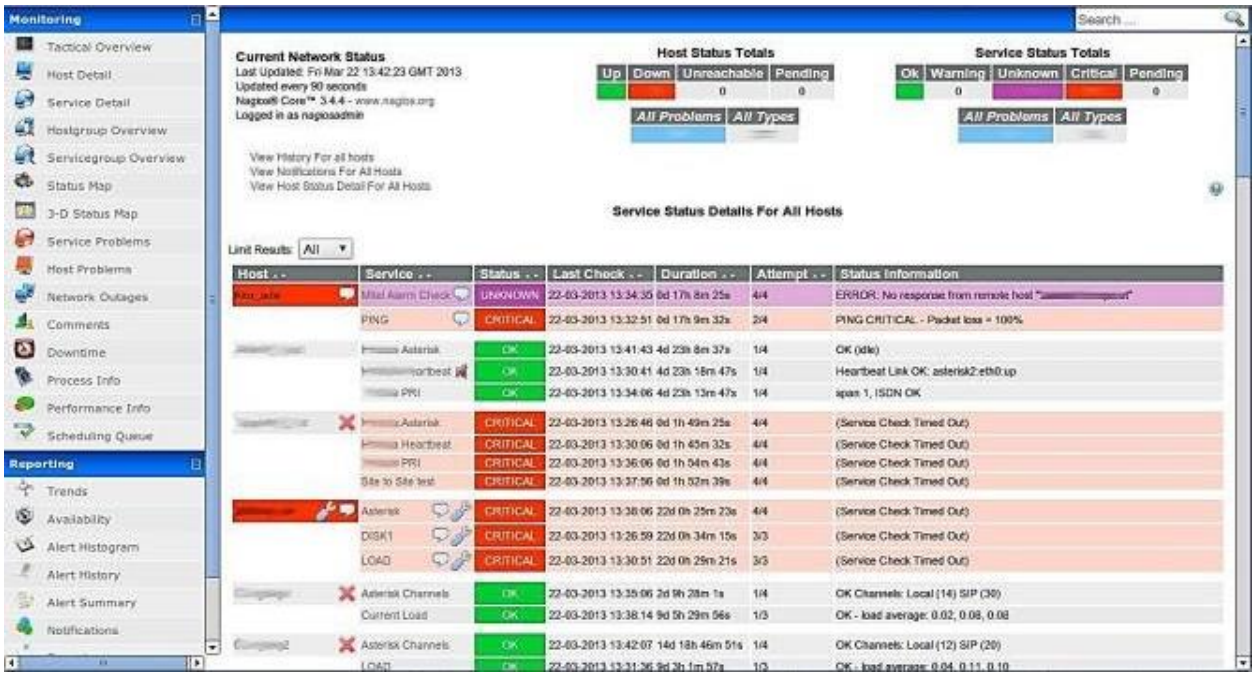


Рис. 1.4. Приклад налаштованого Nagios

ManageEngine Applications Manager[6]**.** Для роботи даного рішення достатньо встановити ManageEngine Applications Manager на центральний сервер та налаштувати його.

Відсутність будь-яких агентів пояснюється тим, що даний продукт використовує існуючі вбудовані засоби та протоколи для моніторингу систем. Прикладами є протоколи WMI (Windows Management Instrumentation) та SNMP (Simple Network Management Protocol). Такий підхід значно спрощує налаштування системи моніторингу і додаткове налаштування розподілених серверів не потрібне.

Проте із-за такого простого налаштування без агентів виникає значний мінус такого підходу – інформація, яка збирається, значно обмежена

загальними метриками, які підтримуються вбудованими засобами для моніторингу.

Як видно з прикладу (рис. 1.5), інформації для візуалізації не так багато.

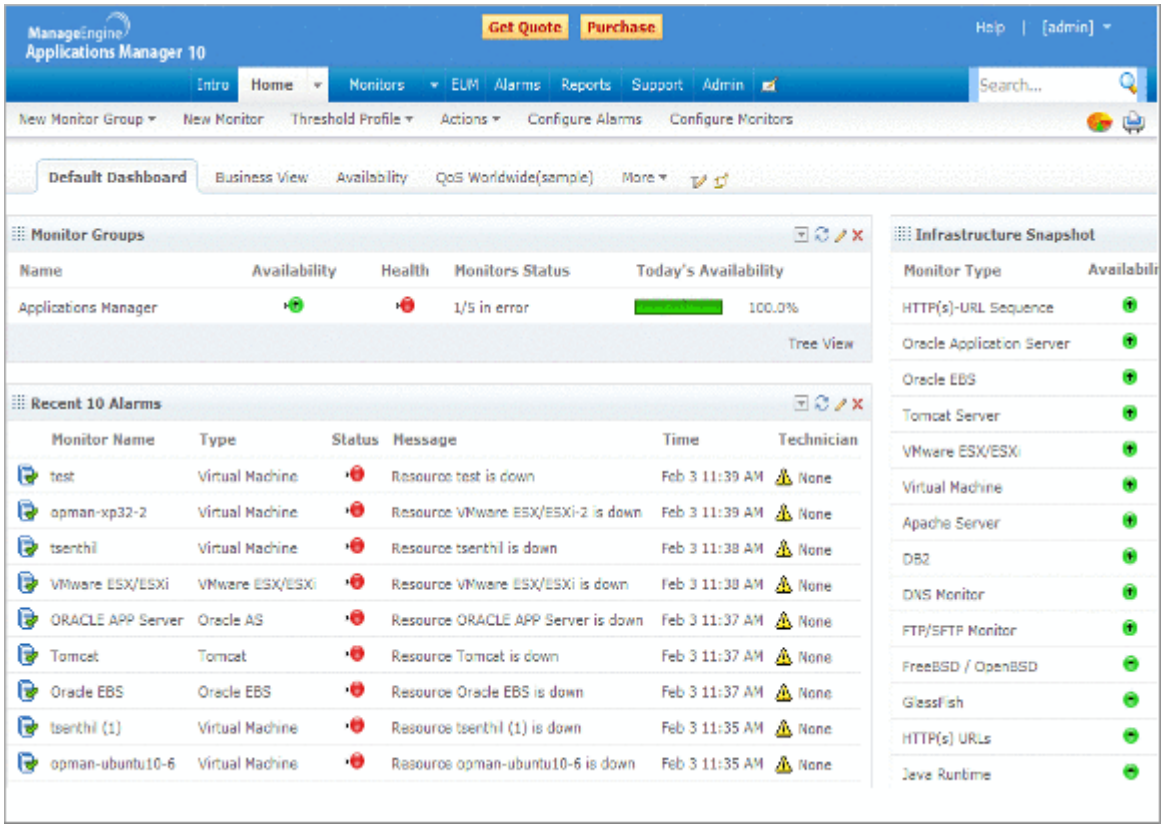


Рис. 1.5. Приклад налаштованого ManageEngine Applications Manager

Основна архітектура **IBM SmartCloud Monitoring**[7] складається з наступних компонентів:

- Enterprise Monitoring Server – основний компонент архітектури, від якого залежать всі інші компоненти;
- Enterprise Portal Client – інтерфейс для перегляду даних, зібраних під час моніторингу;
- Data Warehouse – база даних, де зберігаються дані;
- Enterprise Portal Server – керує різними засобами доступу до даних;
- Агенти – встановлюються та налаштовуються на розподілених серверах.

Існує також велика кількість додаткових компонентів, які не є обов'язковими та встановлюються під окремі задачі.

Для використання даного рішення необхідно не просто налаштувати агентів на кожному сервері, а попередньо розібратися в архітектурі IBM SmartCloud Monitoring (рис. 1.6) [8], оскільки вище наведений лише дуже спрощений опис і без більш детального розгляду дану систему буде важко або навіть неможливо налаштувати.

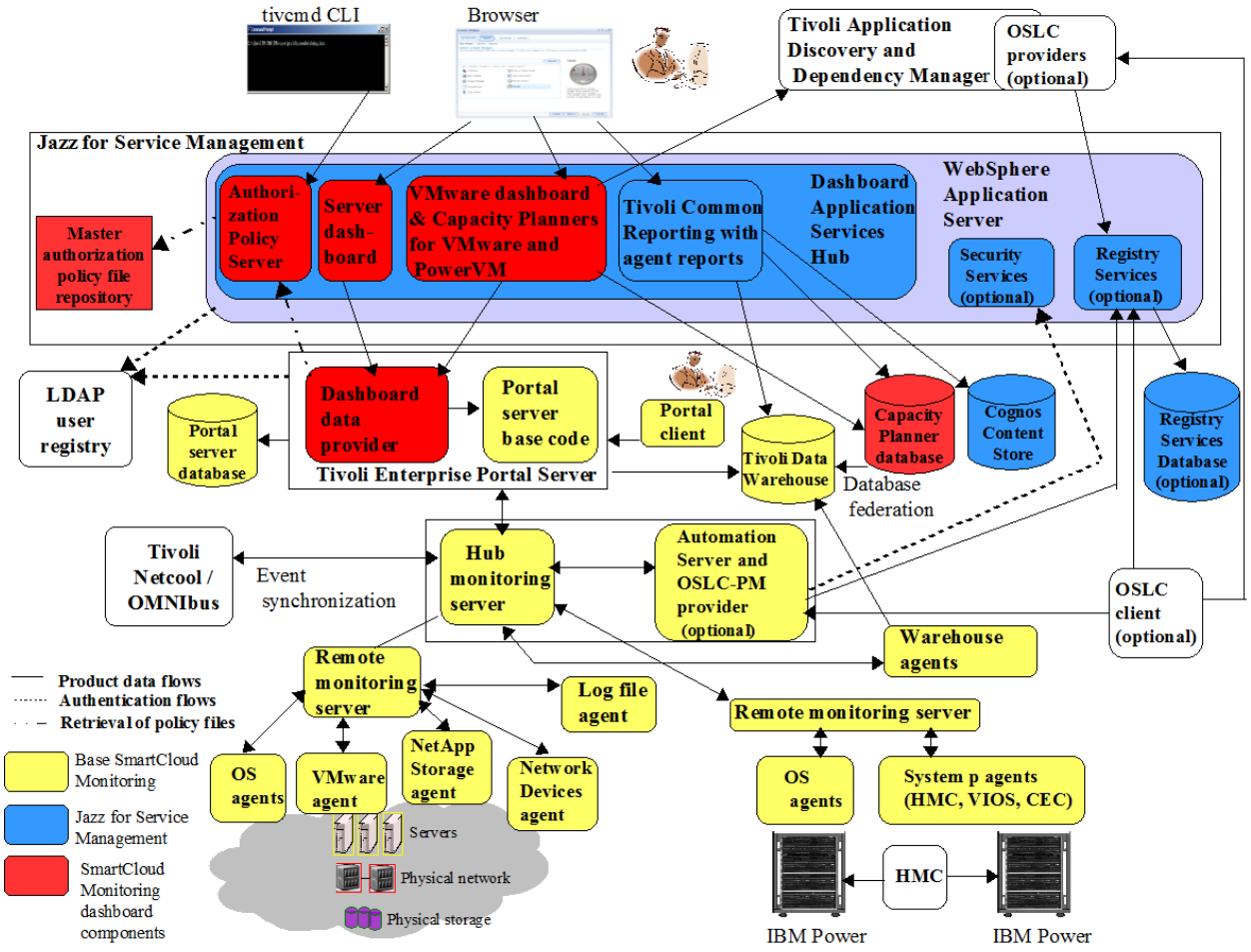


Рис. 1.6. Архітектура IBM SmartCloud Monitoring

Отже, вибір даного рішення тільки ускладнює налаштування моніторингу, що віддаляє нас від основної цілі цієї роботи.

Стек TICK[9]. Цей стек складається з чотирьох компонентів, перші літери яких утворюють назву стеку:

- Telegraf – агент, до якого підключаються необхідні плагіни для збору відповідної інформації;

- InfluxDB – база даних, розроблена для того, щоб працювати з великою кількістю запитів про запис та читання даних, які мають мітку часу;
- Chronograf – графічний інтерфейс для візуалізації даних моніторингу;
- Kapacitor – даний компонент відповідає за сповіщення про знайдені аномалії у зібраних даних.

Налаштування спрощується тим, що при встановленні Telegraf він вже має налаштування за замовчуванням, проте при необхідності все ще необхідно конфігурувати його відповідно до поставлених вимог системи моніторингу.

Структура даного стеку зображена на рис. 1.7.

З даної структури видно, що вона схожа на структуру стеку ELK, де Kapacitor є аналогом Logstash та бітів (англ. beats), Chronograf – Kibana, InfluxDB – Elasticsearch.

Аналогом Telegraf є додаткові плагіни для повідомлень, які встановлюються до стеку ELK при необхідності.

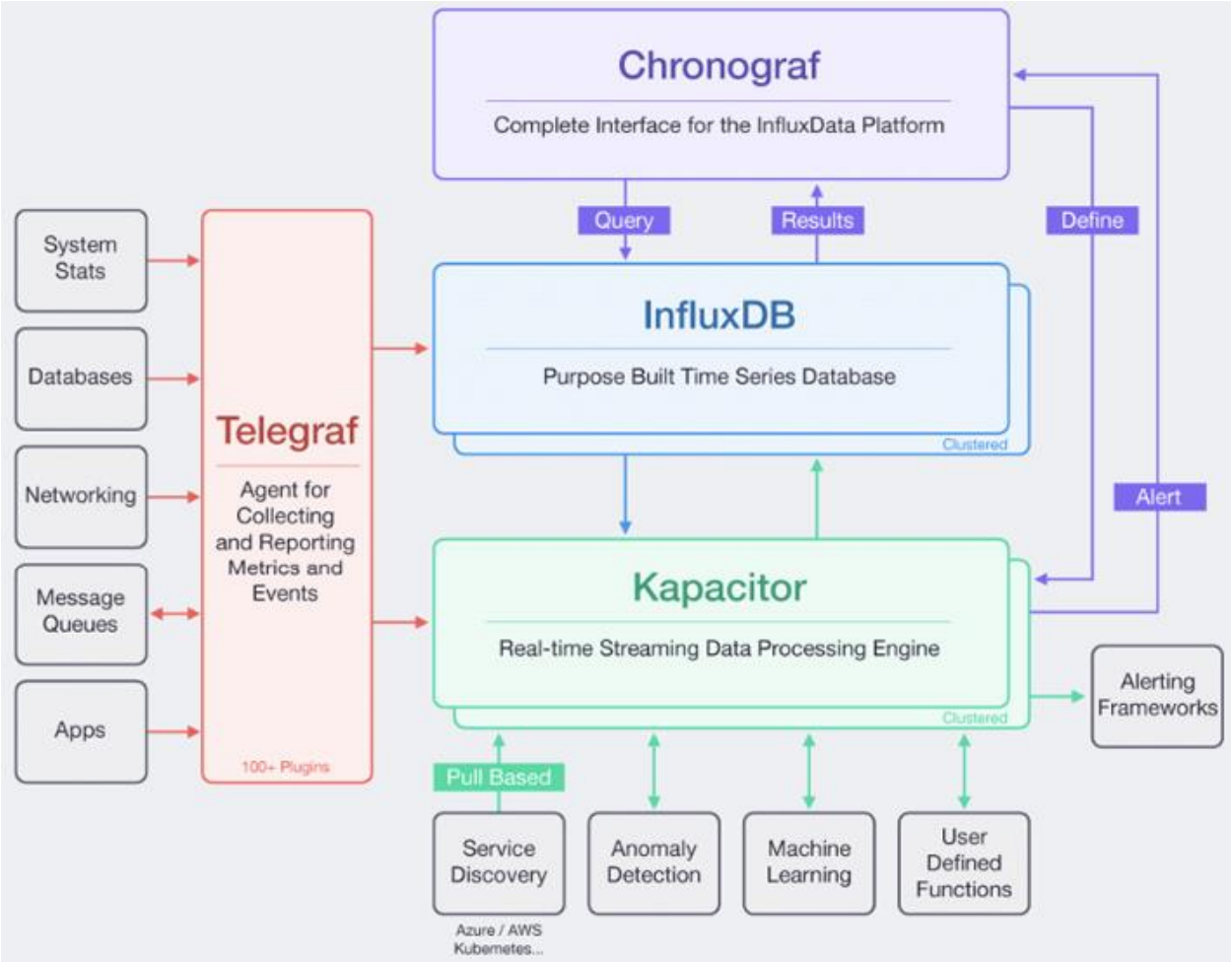


Рис. 1.7. Структура стеку TICK

У таблиці 1.1 наведено порівняльну таблицю описаних рішень для моніторингу.

Табл. 1.1. Порівняльна таблиця рішень для моніторингу

Рішення для моніторингу	**Агенти для збору інформації**	**Автоматизація налаштування**
Стек ELK	Біти (beats), які займають мало ресурсів. Існує великий вибір для різного роду інформації	Відсутня
Zabbix	Агент від Zabbix, можна встановлювати додаткові модулі для збирання різного роду інформації	Присутня, проте все ще необхідно попередньо встановити додаткові компоненти на сервери

Продовження Табл. 1.1. Порівняльна таблиця рішень для моніторингу

Рішення для моніторингу	**Агенти для збору інформації**	**Автоматизація налаштування**
Nagios	Nagios агент, має графічний інтерфейс та можливість встановлення плагінів для різної інформації	Частково (в однорідній системі можна легко використовувати однакові налаштування на різних серверах)
ManageEngine Applications Manager	Відсутні, використовуються вбудовані засоби моніторингу. Інформація, яка збирається, обмежена	Не потрібна
IBM SmartCloud Monitoring	Для кожного виду інформації існує окремий агент	Присутня у невеликої частини агентів[10], проте налаштування всієї системи значно ускладнене архітектурою даного рішення
Стек TICK	Telegraf, до якого можна підключити різні плагіни для збору різної інформації	Відсутня

1.3. Автоматизація налаштування

Як видно, розглянуті рішення (окрім ManageEngine Applications Manager) мають одну спільну проблему – оператору необхідно витрачати свій час на налаштування кожного серверу під задачу моніторингу. Ці

налаштування часто дуже подібні і оператору приходиться виконувати монотонну роботу.

ManageEngine Applications Manager, як описано у попередньому підрозділі, має іншу проблему – він дуже обмежений в інформації, яку він може збирати.

Рішенням проблеми монотонної роботи може бути автоматизація цієї роботи. Автоматизація – це технологія, за допомогою якої процес або процедура виконується з мінімальним сприянням людини [11]. Тобто від людини необхідні лише вказівки про те, що і як треба зробити, а все інше виконується автоматично. Таким чином можна прибрати необхідність виконувати однакову роботу повторно людиною.

В контексті моніторингу, від людини-оператора необхідно лише вказати як саме необхідно налаштувати моніторинг, а саме налаштування виконається програмою. При цьому ці вказівки задаються з одного комп'ютера (центрального сервера), що прибирає необхідність надавати ці вказівки окремо кожному серверу. Також ці вказівки необхідно мінімізувати, тобто якщо конфігурації різних серверів однакові або дуже схожі, то необхідно надати можливість дати одну вказівку, яка застосовується до декількох серверів. Це прибере необхідність повторювати одну і ту ж саму вказівку для кожного сервера і спростить налаштування великої розподіленої системи.

На рис. 1.8 зображено діаграму послідовності при відсутності автоматизації налаштування. Як видно з даної діаграми, оператору необхідно звертатись до кожного серверу та налаштовувати компоненти на ньому окремо. На рис. 1.9 зображено таку ж діаграму послідовності, але з автоматизацією налаштування. Як видно, оператору достатньо звернутися лише до одного сервера і за один раз вказати всі необхідні налаштування. Тоді центральний сервер сам виконає всі налаштування у себе та на вузлах розподіленої системи.

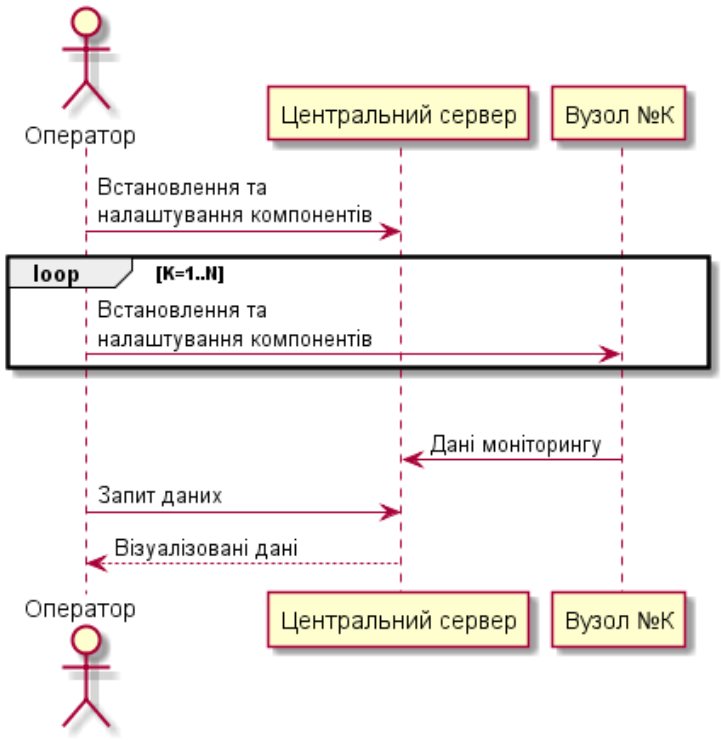


Рис. 1.8. Діаграма послідовності взаємодії оператора з розподіленою системою з ручним налаштуванням моніторингу

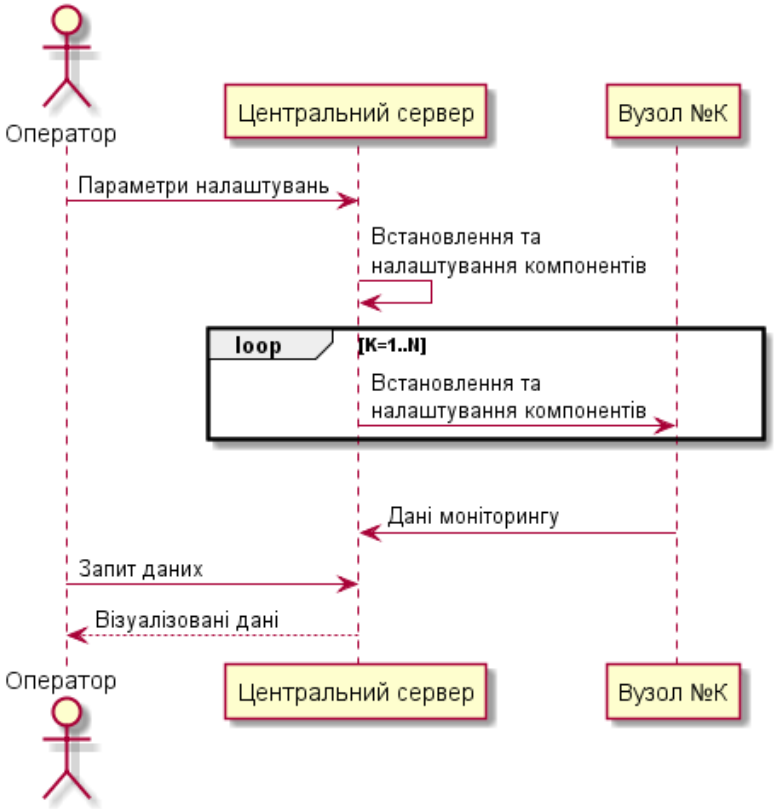


Рис. 1.9. Діаграма послідовності взаємодії оператора з розподіленою системою з автоматизованим налаштуванням моніторингу

Деякі рішення моніторингу дозволяють автоматизувати їх налаштування в розподіленій інформаційній системі. Як описано вище, Zabbix дозволяє автоматизувати налаштування, проте на вузлах розподіленої системи для цього повинні стояти компоненти Zabbix, які це дозволяють. Тобто необхідно або встановлювати операційну систему від Zabbix з даними компонентами, або встановлювати ці компоненти самостійно.

Є рішення, у яких відсутня необхідність автоматизації такого налаштування, оскільки вони користуються інструментами, які є вбудовані у систему. Прикладом є ManageEngine Applications Manager, проте вище описано іншу проблему такого підходу – обмежений вид інформації, яку можна збирати.

Наразі адміністратори часто автоматизують процес написанням спеціальних скриптів, які виконують це налаштування. Проте для спрощення ці скрипти пишуться під систему, яка налаштовується і для повторного використання їх необхідно модифікувати або писати заново.[12]

Отже, новий інструмент автоматизації повинен задовольняти наступним вимогам:

- Автоматизоване налаштування моніторингу може виконуватись у вже існуючій розподіленій системі без втручання переривання задач, на які система вже налаштована. Тобто не треба зупиняти сервер під час налаштування або взагалі встановлювати нову операційну систему.
- Налаштована система моніторингу повинна дозволяти слідкувати за широким спектром видів інформації, яка збирається. Прикладами є файли-журнали, метрики, тощо.
- Інструмент повинен бути універсальним щодо різноманітності можливих розподілених систем. Тобто відсутня необхідність модифікувати код інструменту для різних розподілених систем, всі параметри задаються через його інтерфейс.

- Відсутня необхідність оператора звертатися до кожного вузла розподіленої системи і налаштовувати необхідні компоненти системи моніторингу або компоненти інструменту для автоматизації самостійно.

У даній роботі розробляється модуль, який відповідає цим критеріям. Винятком може бути останній критерій у випадку, коли не налаштований Telnet[13,14]. Проте його налаштування простіше ніж налаштування компонентів системи моніторингу та часто він вже є налаштованим.

Це і є перевагою над автоматизацією від Zabbix – компоненти якого необхідно встановлювати окремо для автоматизації подальшого налаштування, тоді як Telnet часто вже є налаштованими. Навіть якщо лише частина серверів налаштована на Telnet, то окремо налаштовувати кожен сервер необхідно лише для серверів, на яких Telnet не налаштований.

ВИСНОВКИ ДО РОЗДІЛУ 1

З описаної вище інформації можна зробити висновок, що моніторинг розподілених систем – актуальна задача. Налаштування моніторингу може займати багато часу оператора, отже необхідно автоматизувати цей процес налаштування.

При цьому під цією автоматизацією мається на увазі можливість задати необхідні параметри налаштування на одному (центральному) сервері, а потім програма без втручання людини налаштує всі сервери, включаючи центральний.

РОЗДІЛ 2.
АРХІТЕКТУРА МОДУЛЯ МОНІТОРИНГУ

2.1. Вибір інструменту моніторингу для автоматизації його налаштування

Оскільки створити універсальну програму для автоматизації налаштування для всіх рішень моніторингу практично неможливо, то необхідно обрати одне з них та автоматизувати його.

Для цієї задачі було обрано стек ELK з одночасним використанням бітів (англ. beats) та Logstash. Причини вибору даного стеку описані нижче в наступних пунктах:

- Всі компоненти налаштовуються схожим чином через конфігураційні файли на мові YAML. Схожість конфігураційних файлів компонентів значно спрощує написання коду для їх генерації.
- Біти (англ. beats) займають мало ресурсів на вузлах розподіленої системи та є широкий вибір для різних цілей (моніторинг працездатності сервера, метрик, логів і т.д.).
- Окрім різних цілей підтримуються також різні операційні системи.
- Не дивлячись на те, що біти (англ. beats) використовують мало ресурсів і тому обмежені в можливостях, Logstash на центральному сервері надає необхідні засоби для приведення "сирих" даних у потрібний вигляд.

На рис. 2.1 відображено діаграму послідовностей при взаємодії компонентів стеку та оператора зі стеком.

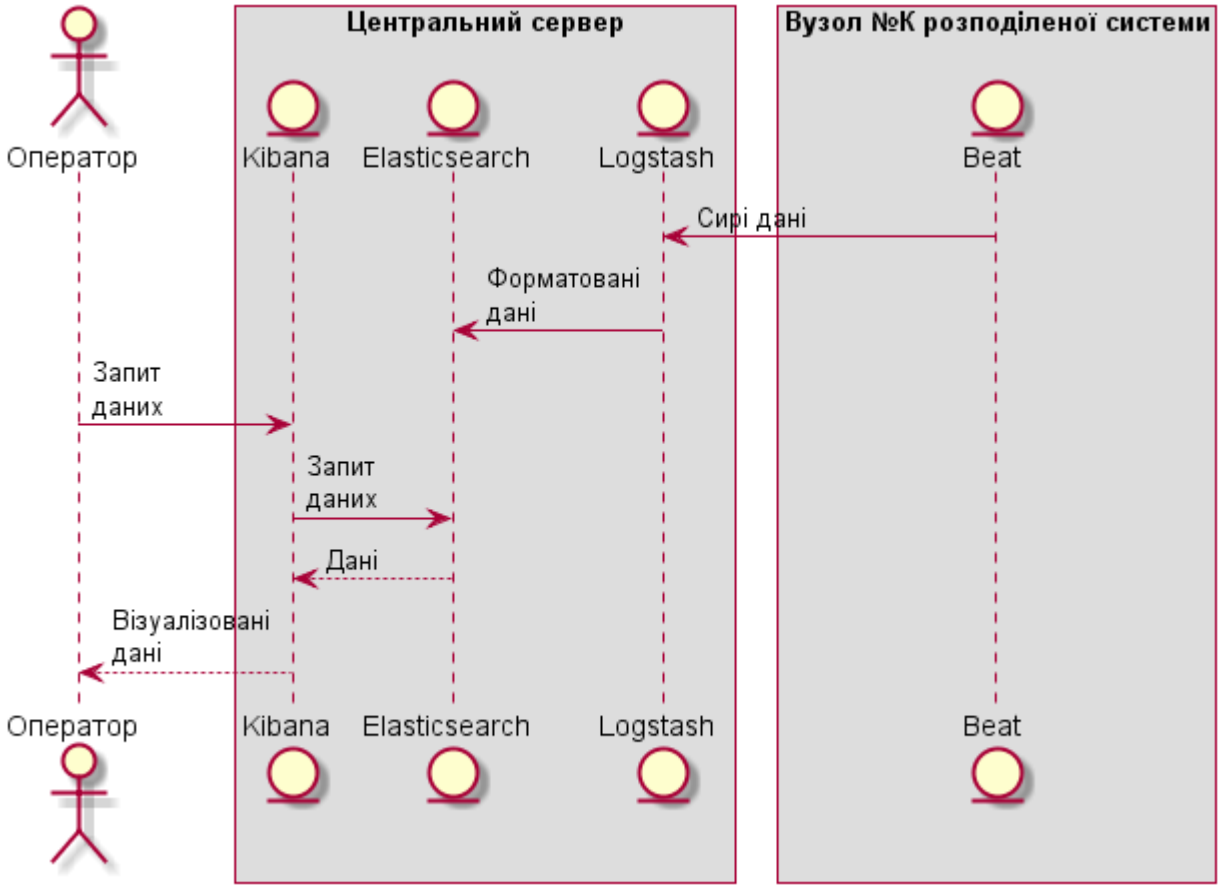


Рис. 2.1. Діаграма послідовності взаємодії компонентів стеку ELK

2.2. Опис можливостей користувача

Оператор буде працювати лише з центральним сервером. Він повинен мати можливість ввести всі необхідні параметри налаштування системи моніторингу на одному (центральному) сервері. На рис. 2.2 відображена діаграма використання модуля оператором.

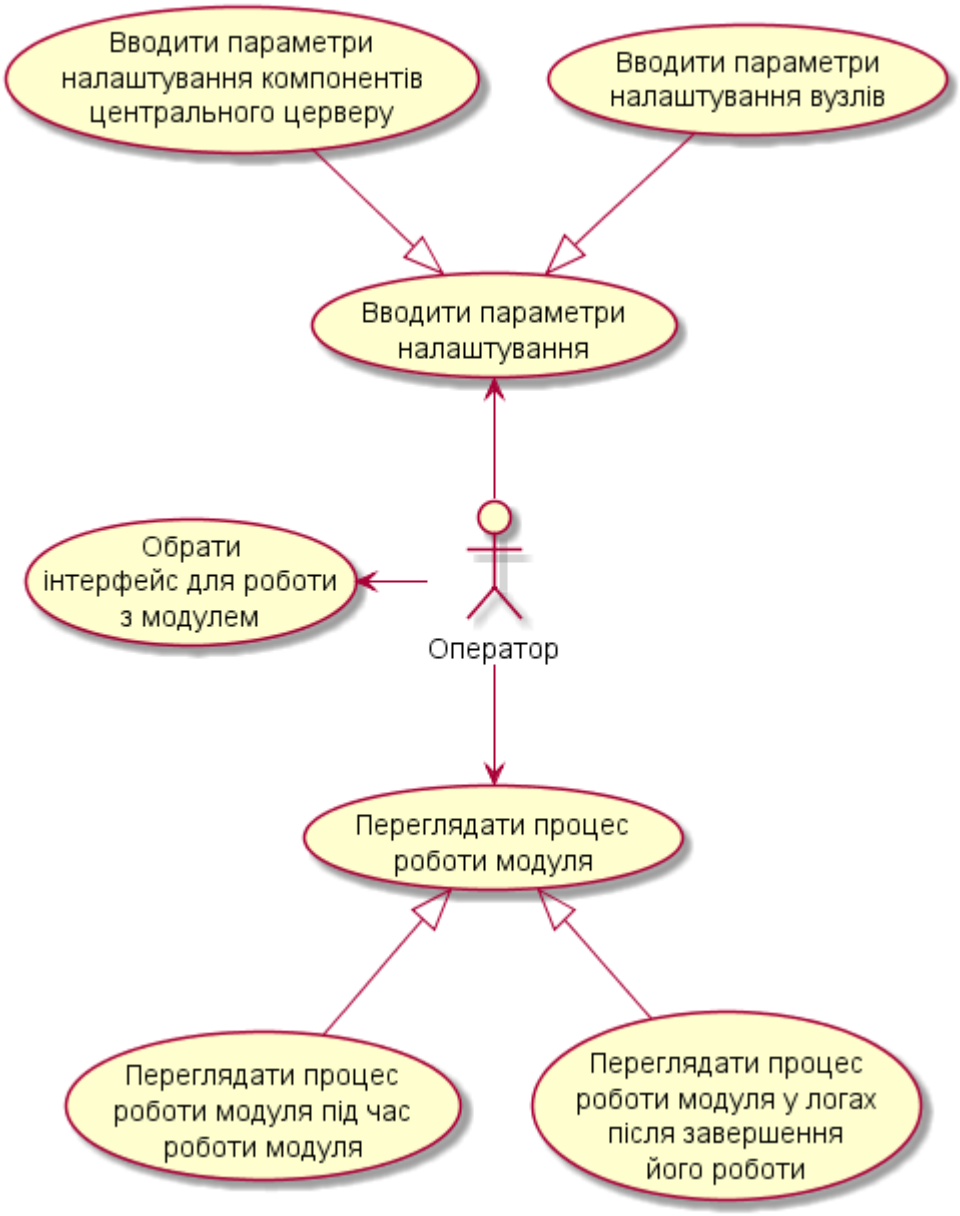


Рис. 2.2 Діаграма використання модуля

Параметри налаштування включають до себе параметри налаштування компонентів центрального серверу, та параметри налаштування бітів (англ. beats) на вузлах розподіленої системи.

Введення параметрів налаштування центрального сервера оператору зазвичай буде не потрібно. По-перше, налаштувань Elasticsearch та Kibana за замовчуванням зазвичай достатньо для цілей моніторингу. По-друге, налаштування взаємодії Logstash з Elasticsearch виконуються автоматично модулем, який розробляється. Тобто зазвичай оператору необхідно вводити ці параметри, тоді як в автоматизованому режимі ці параметри

встановлюються модулем без додаткових вказівок оператора. Модуль все ще повинен надавати можливість вводити ці налаштування при необхідності.

Для вузлів розподіленої системи все ж необхідно ввести інформацію, щоб модуль "знав", за чим повинна слідкувати система моніторингу. Поки інформацію про центральний сервер (таку, як його IP адреси у мережах, до яких він підключений) модуль може встановити сам, інформацію для доступу до вузлів все ще необхідно вводити оператору. Отже, оператор повинен ввести наступні параметри:

- Перелік вузлів розподіленої системи та яким чином до них доступитись (IP адреса, який протокол використовувати, дані для авторизації). Модулем надається можливість задати діапазон адрес, які модуль просканує командою ping і додасть ті з них, які відповіли на цей запит.
- Який тип інформації повинна збирати система моніторингу. Прикладами є журнал-файли, доступність сервера, завантаженість ресурсів. Відповідно до цього, модуль обере які біти (англ. beats) необхідно встановити на вузли розподіленої системи. Наприклад, якщо оператор обрав, що на деякому сервері необхідно буде відслідковувати файл, то модуль встановить на цей сервер Filebeat.
- При необхідності, надати додаткову інформацію. На прикладі того ж Filebeat, якщо оператор задав відслідковування файлів, то необхідно вказати які самі файли необхідно відслідковувати. Для перевірки працездатності сервера можна вказати інтервал відправки повідомлення від сервера, що він працює.
- Також при необхідності, оператор може вказати додаткові параметри для Logstash. Хоча даний компонент знаходиться на центральному сервері, налаштовується він під окремий біт (англ. beat) окремого серверу. Найчастіше це необхідно для

відслідковування файлів-журналів, так як Filebeat відслідковує лише зміни у файлах, а Logstash приводить їх у необхідний вигляд.

Для роботи з модулем оператору надається можливість вибору серед двох інтерфейсів: графічний, та через командний рядок.

Також у оператора є можливість переглядати процес роботи модуля під час або після завершення його роботи. У першому випадку модуль через інтерфейс відображає що він зараз робить, а для другого випадку одночасно записує цю інформацію у файл.

2.3. Робота модуля

Порядок дій модуля наступний:

1. Дочекатися вводу списку вузлів розподіленої системи та відповідних параметрів.
2. Встановити Elasticsearch на центральний сервер. Параметри за замовчуванням залишаються.
3. Встановити Kibana на центральний сервер. Також залишити параметри за замовчуванням.
4. Встановити Logstash на центральний сервер. Вказати Elasticsearch отримувачем даних моніторингу. Якщо були вказані додаткові параметри для перетворення даних, то додати їх до файлу конфігурації.
5. Для кожного сервера встановити необхідні біти (англ. beats) для моніторингу. При наявності додаткових параметрів (таких як, які саме файли відслідковувати), налаштувати їх. У якості отримувача вказати Logstash на центральному сервері (IP адреса центрального сервера у даній мережі та порт, який прослуховує Logstash).

Відповідна діаграма зображена на рис. 2.3.

Рис. 2.3. Алгоритм роботи модуля

На рис. 2.4 відображено діаграму сутностей, з якими модуль буде працювати.

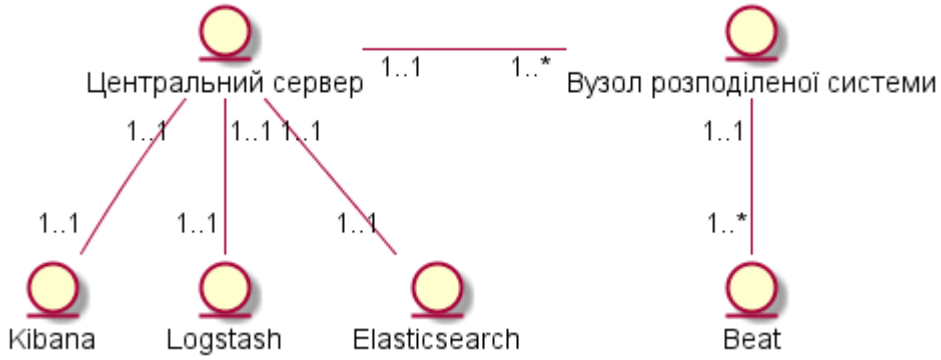


Рис. 2.4. Діаграма сутностей

2.4. Структура модуля

На рисунку 2.5 відображено внутрішню структуру модуля.

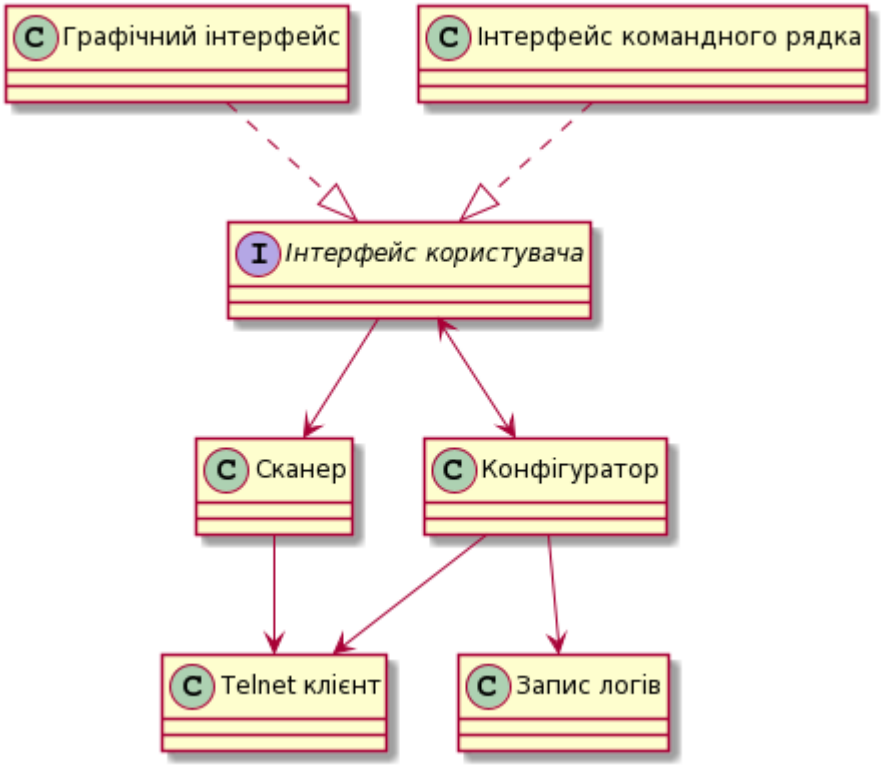


Рис. 2.5. Діаграма класів модуля

Розглянемо компоненти даної структури та їх функції:

- Інтерфейс користувача. Має дві реалізації: графічний інтерфейс та інтерфейс командного рядка. Надає оператору можливість взаємодіяти з модулем, а саме вводити параметри системи моніторингу та переглядати процес роботи модуля під час налаштування.

- Сканер. Надає оператору можливість сканувати діапазон IP адрес командою `ping` [15] та автоматично знаходити відповідні додаткові параметри (наприклад, визначити операційну систему, знайти файли журналів без ручного вводу повного шляху до них, тощо).
- Конфігуратор. Основна частина модуля, відповідає за налаштування системи моніторингу. Отримує параметри через графічний інтерфейс, доступається до вузлів розподіленої системи через протокол Telnet, генерує необхідні команди та файли конфігурацій для вузлів.
- Telnet клієнт. Надає інструменти для доступу до віддалених серверів. Використовується сканером для визначення операційної системи та інших параметрів. Використовується конфігуратором для налаштування вузлів.
- Запис логів. Записує процес налаштування у відповідні файли, які оператор може переглянути після завершення налаштування системи моніторингу.

2.5. Взаємодія компонентів модуля

Висока зв'язність[16,17] компонентів модуля (або любого іншого програмного продукту) ускладнює написання його коду. А саме:

- Зміна коду одного компонента часто призводить до ланцюгової реакції, із-за якої часто приходиться змінювати код інших компонентів.
- Збірка компонентів разом може потребувати більше часу та/або зусиль із-за високої залежності один від одного.
- Окремі компоненти важко тестувати та повторно використовувати в інших проектах із-за великої кількості інших компонентів, від яких залежить тестований або повторно використовуваний компонент.

Отже, необхідно зменшити зв'язність модуля. Для цього існують різні засоби. Розглянемо декілька шаблонів проектування, які дозволяють це зробити:

- Медіатор (англ. Mediator) [18]. Діаграма класів цього шаблону зображено на рис. 2.6. В даному шаблоні компоненти (colleage) звертаються до медіатору, який передає повідомлення відповідним компонентам. При цьому компоненти не знають про один одного і лише знають про медіатор, тоді як медіатор знає про всі компоненти. Тобто медіатор інкапсулює у собі зв'язок між компонентами, а компоненти делегують цю задачу до медіатора.

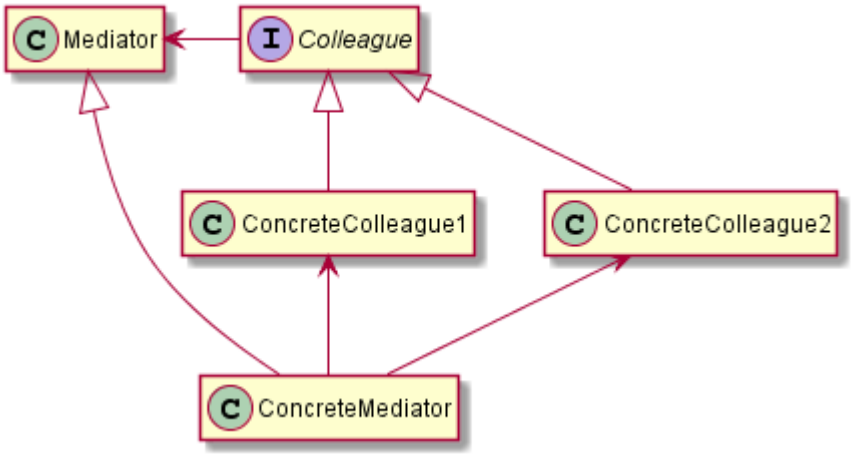


Рис. 2.6. Діаграма класів для шаблону проектування Медіатор

- Спостерігач (англ. Observer) [18]. З рис. 2.7 видно, що спілкування відбувається в одному напрямку. Компоненти знають один про одного, а саме Спостерігачі додають себе до колекції спостерігачів у Суб'єкта через відповідний метод, а Суб'єкт при зміні стану викликає відповідний метод у Спостерігачів, які знаходяться в колекції. Даний шаблон дозволяє зробити залежність один-багато (англ. one-to-many) без високої зв'язності компонентів.

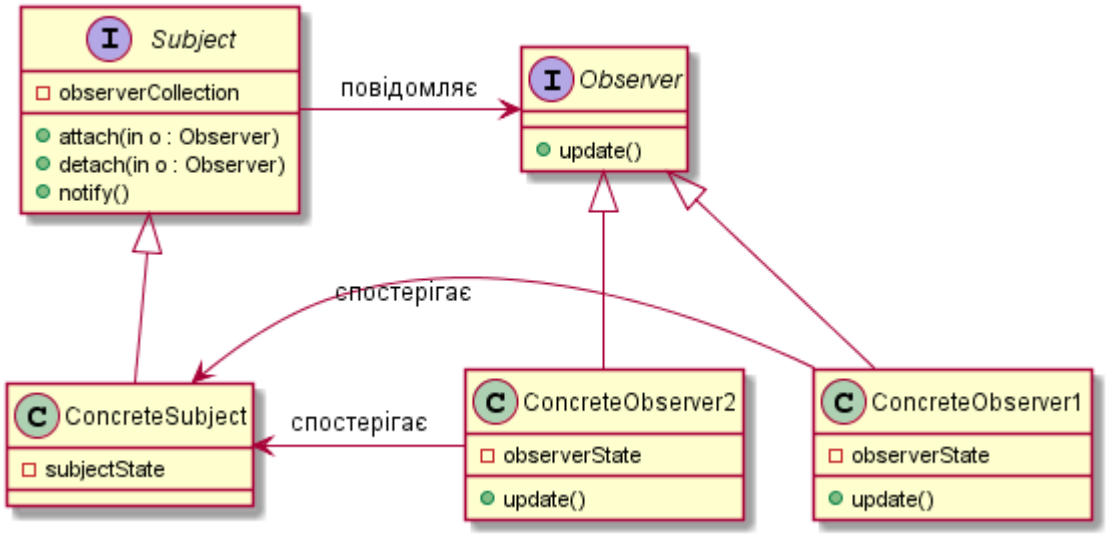


Рис. 2.7. Діаграма класів для шаблону Спостерігач

- Публікація-підписка (англ. Publish-subscribe) [19]. Як видно з рис. 2.8, в даному випадку є теми, на які підписуються Підписники (англ. Subscriber) і є Видавці (англ. Publisher), які видають повідомлення на відповідні теми. Один і той самий компонент може одночасно бути підписником та видавцем. Даний шаблон схожий на шаблон Медіатор тим, що компоненти не знають один про одного, а на шаблон Спостерігач тим, що компоненти самі підписуються на повідомлення.

  Також може бути підписка не по темам, а по контенту. Тобто підписники підписуються на всі повідомлення, а при отриманні повідомлення аналізують його на те, чи потрібно його обробити чи проігнорувати його.

  Також існує гібридна версія, де підписники підписуються на окремі теми та при отримані повідомлення фільтрують їх для себе.

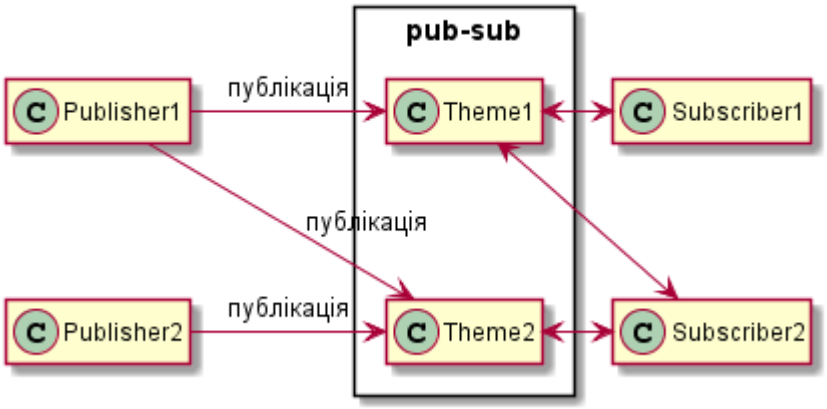


Рис. 2.8. Структура шаблону Публікація-підписка

Останній шаблон (Публікація-підписка) дозволяє зробити компоненти найменш зв'язаними у порівнянні з іншими розглянутими шаблонами (Медіатор та Спостерігач).

На відмінну від шаблону Спостерігач, даний шаблон знімає з відправника обов'язок слідкувати за отримувачами. А на відмінну від Медіатора, дає більшу свободу щодо отримувачів, які можуть отримувати дане повідомлення.

Також він дозволяє згрупувати повідомлення на окремі теми та спрощує реалізацію відправки повідомлень від різних компонентів.

Отже, для даного модуля був обраний шаблон проектування Публікація-підписка з підпискою на теми.

Для реалізації тем повідомлень можна використати шаблон Одинак (англ. Singleton) [18]. Даний шаблон дозволяє отримати максимум один екземпляр деякого класу. Це досягається за допомогою спеціального методу, який при виклику перевіряє чи був створений даний необхідний екземпляр раніше. Якщо ні - то створити його та повернути. Якщо так, то повернути вже створений. У відповідній діаграмі класів (рис. 2.9) відображено лише один клас Singleton, який має даний метод.

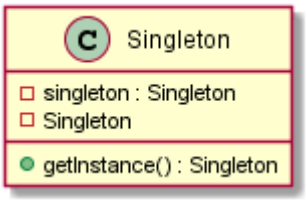


Рис. 2.9. Діаграма класів шаблону Одинак

В даному випадку необхідно створити лише один екземпляр для кожної теми. Замість того, щоб задавати список тем наперед та створювати ці об'єкти при ініціалізації, можна використати вищеописаний шаблон проектування Одинак.

Для цього при кожній зверненні до конкретної теми (публікація або підписка) перевіряється чи існує об'єкт, який відповідає за цю тему. Таким чином для кожної теми буде максимум один відповідний об'єкт, а кількість тем не обмежена.

Таке рішення також має свою назву – шаблон Multiton[20] або колекція Одинаків (рис. 2.10). В даному шаблоні існує приватна колекція, яка зберігає екземпляри відповідно до ключів (або теми у контексті даного модуля). Існує публічний метод, який за заданим ключем (темою) повертає відповідний об'єкт. Таким чином кожному ключу максимум може відповідати один екземпляр.

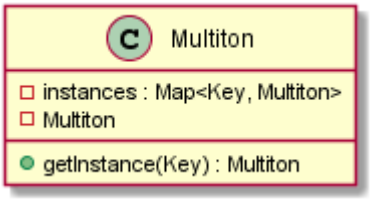


Рис. 2.10. Шаблон Multiton

Як вже було сказано вище, шаблон Публікація-підписка дозволяє компонентам обмінюватись повідомленнями не знаючи один про одного. Таким чином можна, наприклад, створити тему "інтерфейс користувача". Незалежно від того, який саме інтерфейс (графічний чи командний рядок) використовується оператором, компоненти можуть публікувати або підписуватись на повідомлення з даною темою і спілкуватись з компонентом

інтерфейсу не знаючи який саме інтерфейс використовується. Відповідна нова діаграма класів зображена на рис. 2.11.

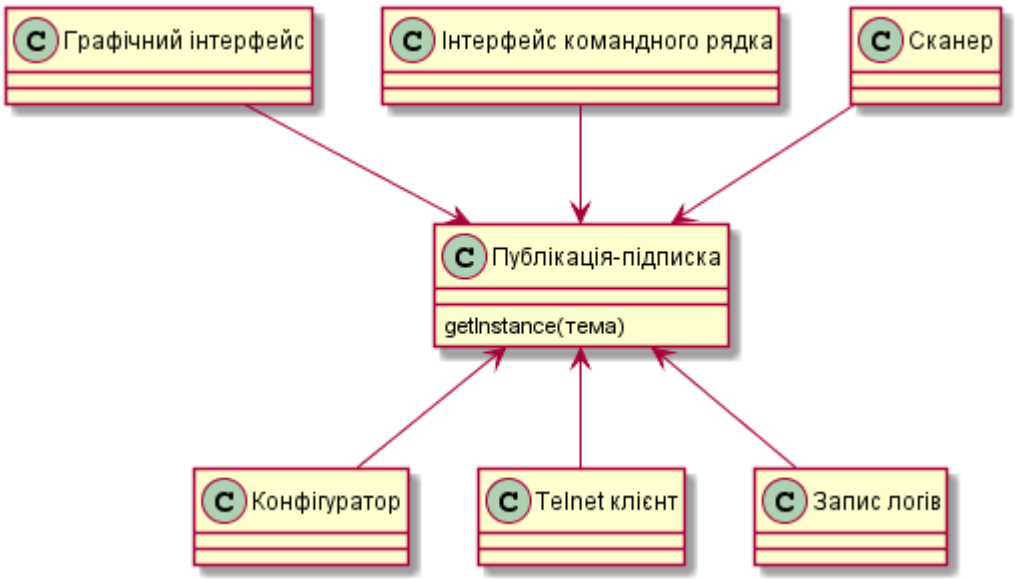


Рис. 2.11. Діаграма класів модуля з шаблоном Публікація-підписка

2.6. Вибір інструментів для написання модуля

Наразі існує широкий вибір інструментів для створення програмних продуктів. Вибір інструменту напряму впливає на складність написання програмного продукту та на можливості даного продукту.

Основним інструментом було обрано платформу Node.js [21]. Причини такого вибору:

- Дана платформа є безкоштовною та з відкритим вихідним кодом. До плюсів відкритого вихідного коду відносяться підвищення надійності та швидкості роботи програми, безпеки даних, тощо. [22, 23]
- Існує широкий вибір допоміжних пакетів для різних цілей, які завантажуються через менеджер npm [24]. Серед них є пакети для спрощення написання програми для роботи з Telnet, взаємодією з користувачем через командний рядок або графічний інтерфейс, тощо.

- Платформа Node.js зараз є дуже популярною та швидко розвивається.

Розглянемо пакети, які можуть спростити написання коду модуля моніторингу. Всі пакети мають відкритий вихідний код, а самі пакети можна знайти у npm.

Почнемо з обробки аргументів при виклику модуля. Деякі популярні пакети, які спрощують реалізацію обробки аргументів для розробника програмного продукту:

- minimist. Даний пакет походить від пакета optimist, який більше не підтримується. minimist дуже простий як у можливостях, так і у використанні. Параметри задаються через '-' або '--'. Приклад параметрів та об'єкта, який отримано після використання пакету minimist:

```
$ node example/parse.js -a beep -b boop
{ _: [], a: 'beep', b: 'boop' }
```

- yargs. Параметри також передаються через '-' або '--'. Є можливість додавання функції валідації та обробки параметрів. Функція обробки приймає параметр від користувача та повертає оброблений параметр. Також є можливість додавання параметрів типів Boolean (розглядається тільки наявність чи відсутність флагу без параметрів), Array (параметри після флагу розглядаються як один масив). Є можливість додавання підкоманд. Автоматична генерація виводу `--help`.
- commander. Надає практично ті самі можливості, що і yarg: валідація і обробка параметрів, різни типи параметрів, під команди, генерація допоміжного меню. Також є можливість вказувати версію, яка буде відображатись при виклику допоміжного меню.

- oclif. На противагу minimist, даний пакет надає дуже широкий інструментарій для обробки аргументів. Практично має все, що і попередні пакети, а також плагіни, защіпки (англ. hooks), автоматичну генерацію інсталятора для того, щоб прибрати необхідність користувачу встановлювати Node.js, тощо.

Для написання даного модуля, використовувати oclif не має необхідності і засобів commander або yarg цілком вистачає. У той же час, minimist навпаки обмежений в можливостях. Оскільки великої різниці між функціоналом commander та yarg немає, то був обраний commander, оскільки він більш популярний.

На відміну від вказування аргументів при виклику модуля, інтерфейс командного рядка дозволяє вказувати всі параметри в інтерактивному режимі. Через аргументи можна передати початкові конфігурації модуля (наприклад, вибір інтерфейсу або вказування файлу готових конфігурацій стеку ELK), тоді як інтерфейс командного рядка допомагає оператору ввести параметри налаштування самої системи моніторингу.

Деякі пакети для інтерфейсу командного рядка:

- inquirer. Надає різни типи питань, які запитуються у користувача: список можливих варіантів, підтвердження дії, пароль (на екрані не відображаються символи, які вводить користувач), текстовий редактор та інші. Параметри питань задаються через об'єкт, який передається функції пакета, а функція повертає проміс (англ. promise), через який програма отримує відповідь користувача. До питань модна додати функції для валідації. Можна встановлювати плагіни для інших типів питань.
- prompts. Схожий на inquirer, має схожі типи питань. Також має декілька типів, які відсутні в inquirer (наприклад, дата). Але не має підтримки плагінів, які дозволяють розширити типи питань.

Було обрано inquirer, оскільки додаткові типи promts в даному модулі використовуватись не будуть, а плагіни в inquirer дозволяють розширити типи при необхідності.

Розглянемо деякі інструменти для створення графічного інтерфейсу:

- Electron. Найбільш популярний інструмент для створення програм з графічним інтерфейсом на Node.js. Він є крос-платформним. Використовує Chromium, тому для створення інтерфейсу використовуються HTML та CSS.
- NodeGui. Даний інструмент використовує Qt, який дозволяє створювати крос-платформні програми з графічним інтерфейсом. Він використовує значно менше пам'яті та ресурсів процесора у порівнянні з Electron, який використовує Chromium. Проте даний інструмент поки що підтримує не всі можливості Qt.
- Vue та express. За допомогою пакета express можна створити веб-сервер, до якого можна доступитись локально через браузер. Пакет Vue допомагає при створенні інтерфейсу в такому рішенні.

Для модуля моніторингу було обрано останній варіант, так як його простіше реалізувати та не потрібно встановлювати додаткових зовнішніх компонентів.

Для сканування діапазону ip адрес є різні засоби:

- Функція exec. Дана функція дозволяє викликати команді операційної системи, приклад використання:

```
var sys = require('sys')
var exec = require('child_process').exec;
function puts(error, stdout, stderr) { sys.puts(stdout) }
exec("ping -c 3 localhost", puts);
```

- Пакет ping. Реалізований за допомогою функції `exec()`, яка описана вище. Прості реалізація та використання.

- Пакет net-ping. Має більше можливостей, у порівнянні з попереднім пакетом (вказування розміру пакету, протоколу, тощо).
- Пакет Node-NMAP, який надає Node.js інтерфейс до сканеру nmap [25]. Даний сканер має дуже широкий вибір параметрів сканування і навіть надає можливість встановити операційну систему віддаленого вузла, що буде корисно при конфігурації стеку ELK.

Під час розробки модуля було помічено, що пакети ping та net-ping працюють дуже повільно, тоді як nmap працює набагато швидше. Це пов'язано з тим, що nmap працює з мережею на низькому рівні, основна частина його коду написана на C та C++. В зв'язку з швидкістю роботи, було обрано даний інструмент для сканування вузлів системи.

Пакети для доступу до вузлів через протокол Telnet:

- telnet-engine. Простий пакет для обміну повідомленнями через протокол telnet.
- telnet-client. Приймає параметри (ip адреса, логін та пароль, тощо) та встановлює зв'язок з віддаленим вузлом. Виконує розпізнавання полей для вводу логіна та пароля. Є найбільш популярним інструментом для Telnet серед пакетів npm.

Оскільки telnet-client спрощує написання коду для зв'язку через Telnet (наприклад, не потрібно аналізувати поля логіну та паролю, оскільки пакет зробить це сам), то він був обраний для цієї роботи.

Пакети для запису логів:

- loglevel. Простий пакет для логів. Виводить їх лише у консоль, має різні рівні важливості повідомлень.
- node-loggly-bulk. Даний пакет використовує Loggly API для запису логів. Логи записуються у форматі JSON замість

звичайного текстового файлу на хмару Loggly, де ці данні можна передивитись у візуалізованому вигляді та проаналізувати.

- winston. Надає широкий функціонал для запису логів. Є вбудовані засоби для запису у файл, консоль, тощо. Також є можливість додавання плагінів для запису у інші засоби (наприклад, Logger).

loglevel є найбільш популярнім, але для цієї роботи не підходить, оскільки не буде можливості переглядати журнал після завершення роботи модуля. node-loggly-bulk не підходить, оскільки він не записує логи у консоль, а також Loggly не завжди буде потрібний. Було обрано winston, оскільки він надає можливість записувати логи одразу у декілька місць.

Інструменти для автоматичної генерації документації коду на основі коментарів у коді у формат HTML:

- Doccо. Дозволяє генерувати документацію для будь-якої мови програмування. Коментарі форматуються за допомогою Markdown. Також є можливість підсвічувати коментарі кольорами. Згенерована документація є відображенням всього коду, де замість коментарів відображається форматований текст.
- groc. Схоже на Doccо, проте форматований текст відображається окремо від коду.
- YUIDoc. Не включає код у документацію. Має теги для позначення типу елемента (модуль, функція, клас, тощо) та теги для параметрів.
- JSDoc. Подібно до YUIDoc має теги, генерує документацію без включення всього коду. Дуже схожий на JavaDoc для мови програмування Java.

Для документування коду модуля моніторингу було обрано JSDoc, так як він має можливість використовувати теги та доволі популярний серед розробників.

Для реалізації шаблону Публікація-підписка використовується вбудований інструмент Node.js – EventEmiter. Даний клас має метод on() для підписки і метод emit() для публікації.

Для організації підписки на різні теми використовується шаблон Multiton, описаний вище. За допомогою цього шаблону для кожної теми може бути максимум один екземпляр класу EventEmiter.

ВИСНОВКИ ДО РОЗДІЛУ 2

Було розглянуто задачу, яку повинен виконувати модуль та його внутрішню структуру.

Було надано діаграму використання (англ. use-case) модуля оператором. Також було надано алгоритм роботи модуля (див. Додаток А). Розглянуто сутності (англ. entity) з якими модуль повинен працювати.

Щодо структури модуля, було розглянуто компоненти, з яких він складається. Можливі рішення для зменшення їх зв'язності (шаблони проектування Медіатор, Спостерігач, Публікація-підписка). Було обрано шаблон проектування Публікація-підписка.

Розглянуті можливі інструменти для реалізації модуля. Було обрано платформу Node.js та набір пакетів npm для реалізації модуля.

РОЗДІЛ 3.
РЕАЛІЗАЦІЯ МОДУЛЯ

3.1. Структура проекту.

Директорія проекту містить наступні основні частини:

- файл `package.json`;
- директорія `bin`;
- директорія `configs`;
- файл `index.js`;
- директорія `src`.

У файлі package.json зберігається інформація щодо проекту. Даний файл використовується менеджером проектів на Node.js – npm. У файлі є інформація про залежності від сторонніх модулів, автор, різні команди для роботи з проектом (наприклад, тестування або запуск проекту) та багато іншої інформації.

У директорії `bin` міститься файл з назвою `monitoring`, у якому вказано, як запускається даний модуль, а саме:

```
#!/usr/bin/env node
require('../index.js')(process.argv);
```

Перший рядок вказує Unix та Unix-подібним операційним системам, що даний модуль необхідно запускати через Node.js, тоді як у операційній системі Windows даний рядок ігнорується і необхідно явно вказувати у якому середовищі необхідно запускати даний файл.

Даний файл створювався для того, щоб додати можливість запускати модуль за допомогою спеціально заданої команди. Для цього у файлі package.json міститься наступний текст:

```
"bin": {
  "monitoring": "bin/monitoring",
  "mntr": "bin/monitoring"
}
```

Після виконання команди `npm link` модуль можна запускати у любій робочій директорії без явного вказування шляху до проекту модуля за допомогою команди `monitoring` або `mntr`.

У директорії `configs` зберігаються файли з параметрами налаштування стеку ELK. Проте оператору не обов'язково вказувати файли с конфігураціями у цій директорії і може використовувати будь-яку директорію за його вибором.

Файл `index.js` обробляє параметри запуску модуля та ініціалізує необхідні його частини с потрібними параметрами. Наприклад, при запуску модуля, оператор може обрати інтерфейс роботи з модулям, тоді даний файл ініціалізує відповідний інтерфейс.

У директорії `src` міститься основна частина коду модуля:

- директорія `gui` – відповідає за графічний інтерфейс роботи з модулем;
- директорія `cli` – відповідає за інтерфейс командного рядка для роботи з модулем;
- директорія `configurator` – ядро модуля, відповідає за налаштування моніторингу розподіленої системи на основі заданих параметрів;
- файл `network/scanner.js` – відповідає за сканування доступних вершин;
- файл `network/telnet.js` – відповідає за комунікацію з вершинами розподіленої системи;
- файл `loger.js` – відповідає за запис процесу до файлу та у консоль;
- файл `publish-subscribe.js` – реалізує шаблон Публікація-підписка, про який описано в попередньому розділі. Використовується для взаємодії компонентів модуля та для меншої залежності їх один від одного.

На рис. 3.1 зображена діаграма класів, де замість концепцій (див. рис. 2.11) відображені дані компоненти.

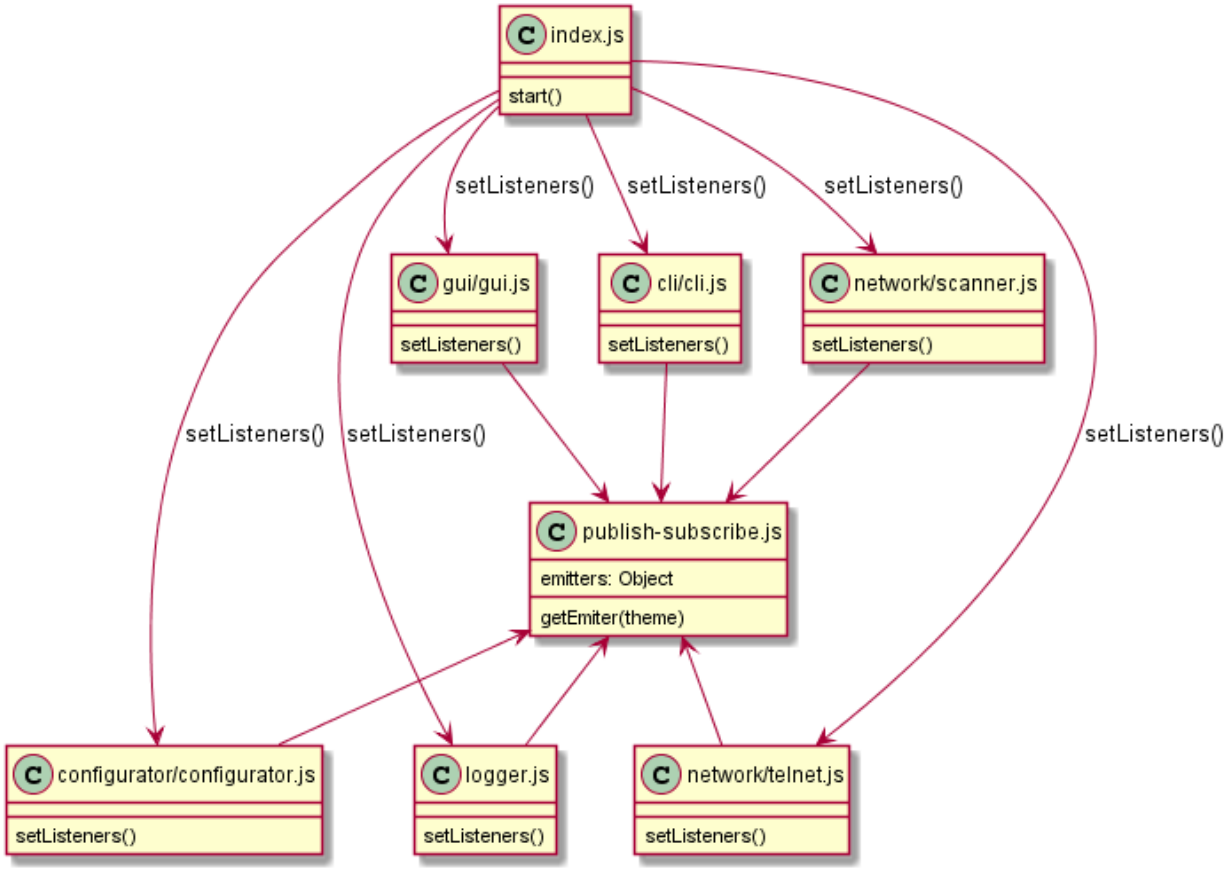


Рис. 3.1. Діаграма класів елементів модуля

3.2. Ініціалізація модуля

Ініціалізація модуля відбувається у файлі `index.js`. А саме, відбувається обробка параметрів запуску модуля та ініціалізація його компонентів.

Для обробки параметрів використовується пакет commander, який виконує цю обробку та повертає об'єкт, в якому строкові параметри перетворені у необхідний вигляд. Також даний пакет автоматично генерує вивід для параметру `--help` та виконує валідацію правильності заданих параметрів. Частина коду, яка відповідає за цю обробку:

```
const program = require('commander');
.................
program
```

```
        .option('-i --interface <type>', 'user interface', 'cli')
        .option('-f --configfile <path>', 'JSON file instead of generating new one');
    program.parse(args);
```

В подальшому доступ до оброблених параметрів відбувається через об'єкт `program`. Наприклад, для отримання шляху до файлу с конфігураціями використовується `program.configfile`, а для доступу для обраного інтерфейсу `program.interface`. В подальшому відбувається обробка цих параметрів.

Для ініціалізації компонентів модуля, вони вказуються на початку даного файлу:

```
const getEmiter = require('./src/publish-subscribe.js');
const gui = require('./src/gui/gui.js');
const cli = require('./src/cli/cli.js');
const configurator=require('./src/configurator/configurator.js');
const telnet = require('./src/network/telnet.js');
const scanner = require('./src/network/scanner.js');
const logger = require('./src/logger.js');
```

Як видно, тут також присутній компонент `publish-subscribe.js`, який використовується для подачі сигналу відповідним компонентам про початок роботи.

Після цього виконується сама ініціалізація деяких компонентів, а саме тих, які необхідно ініціалізовувати при будь-яких параметрах запуску модуля:

```
configurator.setListeners();
telnet.setListeners();
scanner.setListeners();
logger.setListeners();
```

Конфігуратор ініціалізується завжди, оскільки це основна частина модуля і без нього модуль не буде працювати.

Компонент, який відповідає за Телнет протокол також ініціалізується завжди, оскільки конфігуратор не може без нього налаштовувати вершини розподіленої системи. У майбутньому можна розширити функціонал модуля, замінюючи даний компонент на інший, який буде відповідати за інші протоколи зв'язку, наприклад, SSH. Тоді ініціалізація одного с них буде відбуватися вибірково в залежності від параметрів, які були вказані оператором.

Сканнер ініціалізується для випадку, якщо оператор захоче вказати діапазон адрес замість їх списку.

Логгер ініціалізується для запису процесу у файл-журнал та у консоль, він підписується на повідомлення, які буде відправляти конфігуратор у процесі налаштування та записує їх у файл та консоль.

Далі відбувається обробка параметрів та запускається відповідний інтерфейс:

```
const uiEmiter = getEmiter('ui');
var configs;
  if (!program.configfile) {
    switch(program.interface.toLowerCase()) {
      case 'cli':
        configs = cli.setListeners();
        break;
      case 'gui':
        configs = gui.setListeners();
        break;
      default:
        console.log(`Cannot recognize interface '${program.interface}'`);
        return;
    }
    uiEmiter.emit('requestConfig');
  } else {
    try {
```

```
          configs = JSON.parse(fs.readFileSync(program.configfile,
'utf8'));
        } catch(err) {
          console.log('Cannot parse config file:', err);
          return;
        }
        uiEmiter.emit('configs', configs);
      }
```

Як видно, відбувається ініціалізація обраного інтерфейсу. Після чого публікується повідомлення с запитом на генерацію об'єкта з конфігураціями стеку ELK.

Якщо ж було обрано шлях до файлу с конфігураціями замість інтерфейсу, то даний файл обробляється та видається повідомлення с об'єктом с конфігураціями.

3.3. Шаблон Публікація-підписка

Для реалізації даного шаблону використовується вбудований пакет від Node.js – events:

```
const EventEmitter = require('events');
```

Даний компонент містить у собі змінну, яка зберігає емітери (англ. emitter) для кожної теми. При ініціалізації даного компоненту створюється пустий об'єкт:

```
let emiters = {};
```

Назовні експортується наступна функція:

```
function getEmiter(theme) {
  if (!emiters[theme]) emiters[theme] = new Emitter();
  return emiters[theme];
}
module.exports = getEmiter
```

Дана функція приймає тему, на яку треба підписатися або опублікувати.

Після цього перевіряє чи присутня вже ця тема у об'єкті `emiters`. Якщо ні, то створюється новий емітер, який відповідає цій темі. Таким способом шаблон, який описаний у попередньому розділі – колекція Одинаків (англ. Multiton).

Як видно, за допомогою вбудованого емітера у Node.js, реалізувати даний шаблон доволі просто і займає всього декілька рядків коду.

Деякі приклади використання даного компоненту вже було наведено при описі `index.js`. Наведемо повний список використання компоненту `publish-subscribe.js`:

- отримання емітера на обрану тему:
  ```
  const uiEmiter = getEmiter('ui');
  ```
- публікація на обрану тему:
  ```
  uiEmiter.emit('requestConfig');
  ```
- підписка на обрану тему (`createConfigs` – функція, яка викликається для обробки повідомлення):
  ```
  uiEmiter.on('requestConfig', createConfigs);
  ```

3.4. Параметри налаштування стеку ELK

Конфігурація для налаштування задається у форматі JSON. Для цього використовується JSON схема[26]. JSON схема дозволяє розробнику вказати які дані та у якому форматі повинні знаходитись у JSON файлі, в якому зберігаються параметри налаштування модуля моніторингу розподіленої системи.

У JSON файлі с конфігураціями є два основні поля: для налаштування центрального сервера, та для налаштування віддалених серверів.

Формат налаштувань центрального сервера у JSON схемі задається наступним чином:

```
"central": {
      "type": "object",
      "required": [
        "installStack"
```

```
      ],
      "properties": {
        "installStack": {
          "type": "boolean"
        },
        "configLogstash": {
          "type": "object"
        }
      }
    }
```

У ньому є обов'язкове поле `installStack`, яке дозволяє вказати, чи потрібно встановлювати ELK стек на центральний сервер. Можливість відмовитись від його встановлення потрібна для випадків, коли стек на центральному сервері вже встановлений. Це може бути, наприклад, у випадку, коли розподілена система розширюється при вже налаштованому стеку. В такому випадку використовується тільки друга частина налаштувань – налаштування вершин розподіленої системи.

Також серед властивостей налаштувань центрального сервера є налаштування Logstash. В них можна вказати додаткові параметри для перетворення сирих даних у потрібний вигляд перед збереженням їх до Elasticsearch. Це може бути, наприклад, виокремлення дати, тегів, тощо с повідомлення. Це допоможе аналізувати дані. Наприклад, відфільтрувати по тегу, який відповідає за помилки, за вказаний період часу, та побудувати відповідний графік.

Формат налаштувань віддалених серверів дозволяє задавати групи віддалених серверів для випадків, коли їх налаштування однакові:

```
"remote": {
      "type": "array",
      "minItems": 1,
      "items": {
        "type": "object",
```

```
            "required": [
              "group",
              "ip",
              "os",
              "beats"
            ],
            ..............
          }
    }
```

Налаштування віддалених серверів задаються у вигляді масиву таких груп. Для кожної групи вказується її назва, список ip адрес, для яких застосовуються дані налаштування, операційна система даних серверів та які саме біти необхідно встановити.

Назва групи задається типом `string` і використовується виключно для зручності оператора:

```
"group": {
    "type": "string"
}
```

IP адреси задаються масивом, при цьому адреси повинні відповідати правильному формату, для цього задається регулярний вираз. Також масив має містити мінімум одну адресу, інакше дане налаштування немає сенсу. `uniqueItems` використовується для заборони однакових адрес, це допомагає знаходити ситуації, коли оператор випадково вказав одну й ту саму адресу двічі. Схема для ip адрес:

```
"ip": {
            "type": "array",
            "minItems": 1,
            "items": {
              "type": "string",
              "pattern":     "^(?:(?:25[0-5]|2[0-4][0-9]|[01]?[0-
9][0-9]?)\\.){3}(?:25[0-5]|2[0-4][0-9]|[01]?[0-9][0-9]?)$"
            },
```

```
            "uniqueItems": true
    }
```

Назва операційної системи використовується для визначення формату команд, які необхідно передавати при настройці бітів (англ. beats).

Операційні системи обмежені до тих, які підтримуються модулем, що розробляється:

```
    "os": {
            "type": "string",
            "enum": ["linux", "windows"]
    }
```

Для бітів використовується тип `object`. Причиною вибору об'єкта замість масиву є той факт, що `uniqueItems` у масивів застосовується тільки для простих даних. У нашому випадку необхідно переконатися, що біти (англ. beats) не повторюються, але самі біти повинні бути представленні складними об'єктами. Для забезпечення унікальності, назва бітів використовується у якості назви відповідного поля, які завжди повинні бути унікальні, тому і використовується об'єкт для описання бітів.

Розглянемо формат налаштувань деяких з них.

Heartbeat дозволяє переконатися у доступності сервера. Обов'язковим параметром для нього є `timeGap`, який вказує проміжок часу, через який необхідно перевіряти доступність.

```
    "heartbeat": {
                "type": "object",
                "required": [
                  "timeGap"
                ],
                "properties": {
                  "timeGap": {
                    "type": "integer"
                  }
                }
    },
```

Filebeat дозволяє слідкувати за файлами. Найчастіше використовується для слідкування за файлами-журналами. В його налаштування входить список файлів, за якими необхідно слідкувати. Мінімальна кількість файлів дорівнює одному, адже інакше даний біт немає необхідності встановлювати. Відповідна частина схеми:

```
"filebeat": {
                "type": "object",
                "required": [
                  "files"
                ],
                "properties": {
                  "files": {
                    "type": "array",
                    "items": {
                      "type": "string"
                    },
                    "minItems": 1
                  }
                }
},
```

Відповідно у налаштуваннях також присутні деякі інші біти, а при необхідності цей список легко розширювати налаштуваннями інших бітів (не забуваючи про додавання їх обробки до основного коду).

На основі цих схем генеруються відповідні графічний інтерфейс або інтерфейс командного рядка. Також є можливість вказувати шлях до файлу, у якому знаходяться вже готові налаштування. Він перевіряється вищезазначеною схемою на правильність його формату.

3.5. Графічний інтерфейс.

Для графічного інтерфейсу використовується json-editor[27]. Даний інструмент дозволяє на основі JSON схеми згенерувати відповідну веб-сторінку, на якій можна ввести дані відповідно до схеми.

Створення сторінки відбувається на стороні клієнта (браузера), для цього з сервера надаються шаблон сторінки `config-page.html`, скрипт `json-editor.js` , схема `configs-schema.json` та основний клієнтський скрипт `main.js`. У файлі main.js підключаються дані ресурси та створюється редактор об'єкту на основі схеми:

```
const editor =
    new JSONEditor(document.getElementById('editor_holder'),{
       schema: schema
    });
```

Під час підтвердження вводу даних відбувається перевірка відповідності згенерованого об'єкту до схеми і відправляється назад до серверної частини програми:

```
if (editor.validate().length) {
  resultElement.innerHTML = 'Correct errors before submiting';
} else {
  resultElement.innerHTML = 'Submitted!'
  socket.emit('submit', editor.getValue());
}
```

На рис. 3.2 відображено приклад заповнення згенероаної сторінки з відображенням помилок при неправильному вводі параметрів.

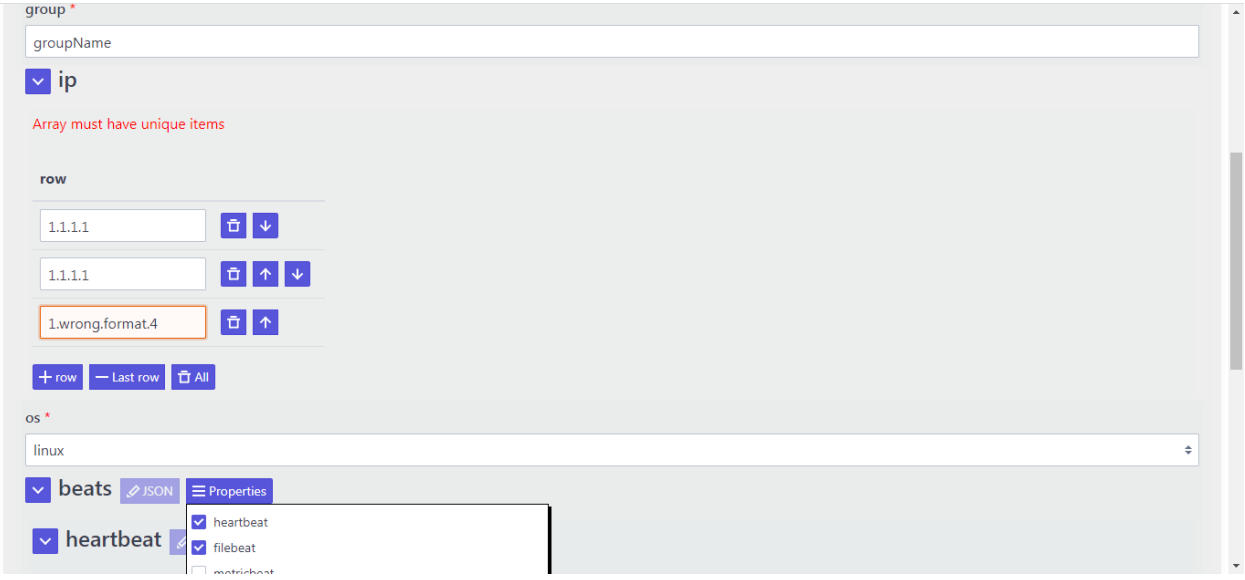


Рис. 3.2. Приклад згенерованої сторінки з відображенням помилок

3.6. Інтерфейс командного рядка

Для реалізації інтерфейсу командного рядка використовується пакет inquirer, який дозволяє задавати у користувача питання у різних форматах та перевіряти їх правильність. Приклад роботи зображено на рис. 3.3, частина коду з прикладом використанням пакета використання наведена нижче.

```
inquirer.prompt([{
                type: 'list',
                message: 'How do you want to add IPs?',
                choices: [
                  'Input IP by hand',
                  ..........
                ]
}]).then(async function (answers) {.........});
```

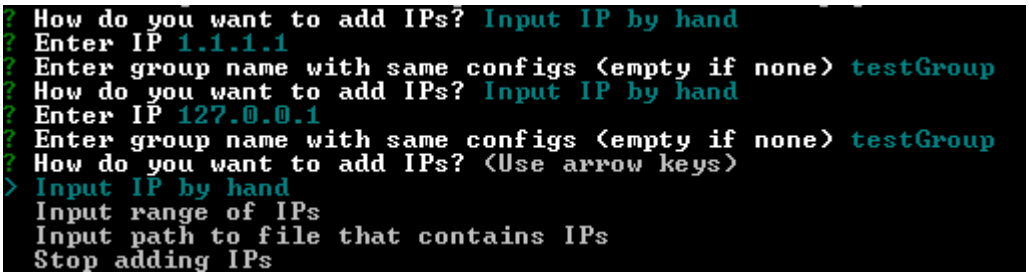


Рис. 3.3. Приклад роботи з інтерфейсом командного рядка

3.7. Конфігуратор

Даний компонент модуля є його основною частиною. У ньому відбувається налаштування моніторингу на основі вказаних параметрів.

Назовні експортується функція `setListeners()` для ініціалізації даного компонента. У процесі ініціалізації створюється підписка на повідомлення з конфігураціями від інтерфейсу користувача:

```
function setListeners() {
  uiEmiter.on('configs', initConfig);
}
```

Функція `initConfig()` обробляє задані параметри налаштування моніторингу. У першу чергу перевіряється правильність введених даних:

```
  const ajv = new Ajv();
```

```
if (!ajv.validate(schema, configs)) {
   console.log('Configs are invalid')
   return;
}
```

Після цього виконується встановлення компонентів центрального сервера. При успішному налагоджені центрального сервера, починається налаштування віддалених серверів:

```
const result = setupCentral(configs.central);
if (result === -1) {
  logEmiter.emit('write', {level:'error', msg:'failed to setup sentral server, see logs for more info'});
  return;
} else {
  setupRemote(configs.remote);
}
```

Для налаштувань у директорії `configurator/scripts` містяться різні скрипти для встановлення, налаштування та запуску компонентів стеку ELK. Для кожного скрипта є дві версії файлу: для Linux та для Windows. Наприклад, наступний скрипт використовується для встановлення компонентів стеку на центральний сервер:

```
wget -qO - https://artifacts.elastic.co/GPG-KEY-elasticsearch | sudo apt-key add -
echo "deb https://artifacts.elastic.co/packages/7.x/apt stable main" | sudo tee -a /etc/apt/sources.list.d/elastic-7.x.list
sudo apt-get update
sudo apt-get install elasticsearch, kibana, logstash
```

Розглянемо окремо функції `setupCentral()` та `setupRemote()`. На початку перевіряється чи хоче користувач встановити компоненти на центральний сервер:

```
function setupCentral(centralConf) {
  if (centralConf.installStack) {
```

Після цього обирається скрипт для поточної операційної системи:

```
switch (os.platform()) {
  case 'win32':
    installCMD = path.join(__dirname, 'scripts',
                           'central', 'instal.cmd');
    break;
  case 'linux':
    installCMD = 'sudo ' + path.join(__dirname, 'scripts',
                                     'central', 'install.sh');
    break;
}
```

Та запускається на виконання:

```
    exec(installCMD, (error, stdout, stderr) => {
      ..... // обробка помилок після інсталювання
      logEmiter.emit('write',   {msg:'installed   components   on
central server', data:stdout, level:'info'});
      configLogstash(centralConf.configLogstash);
    });
  }
}
```

Як видно, тут же запускається налаштування Logstash.

Розглянемо тепер функцію, яка налаштовує віддаленні вузли:

```
function setupRemote(remoteConf) {
  remoteConf.forEach(group => {
    const ext = (group.os === 'linux') ? '.sh' : '.cmd';
    group.ip.forEach(ip => {
      .......
}})})}
```

Як видно, дана функція перебирає кожну ip адресу у кожній групі з однаковими налаштуваннями. У середині даних циклів відбувається генерація скриптів з налаштуванням бітів та відправляється повідомлення з необхідними даними для виконання необхідних скриптів:

```
telnetEmiter.emit('send', {ip:ip, scriptPath:path.join(
                    __dirname, 'scripts', beat, 'install'+ext)});
const confScript = generateConfigBeatScript(
                     beat, group.beats.beat, ext);
telnetEmiter.emit('send', {ip:ip, scriptString: confScript});
telnetEmiter.emit('send', {ip:ip, scriptPath:path.join(
                    __dirname, 'scripts', beat, 'start'+ext)});
```

Сама функція `generateConfigBeatScript()` зчитує шаблон відповідного біта (англ. beat):

```
var script = fs.readFileSync(path.join(
                    __dirname, 'scripts', beat, 'config'+ext));
```

Після чого в залежності від біта підставляє необхідні параметри. Наприклад для heartbeat частина коду виглядає наступним чином:

```
case 'heartbeat':
      script = script.replace('{timeGap}', conf.timeGap);
      break;
```

3.8. Перевірка роботи модуля

Для тестування модуля було створено модель розподіленої системи за допомогою інструменту Docker[28]. Даний інструмент дозволяє зробити це на одному комп'ютері з невеликою затратою ресурсів комп'ютера. Центральним сервером виступала операційна система комп'ютера, тоді як вершини розподіленої системи моделювались у Docker.

Приклад роботи інтерфейсу наведено вище на рисунках 3.2 та 3.3.

Приклад роботи Kibana, де відображаються зібрані дані наведено на рис. 3.4.

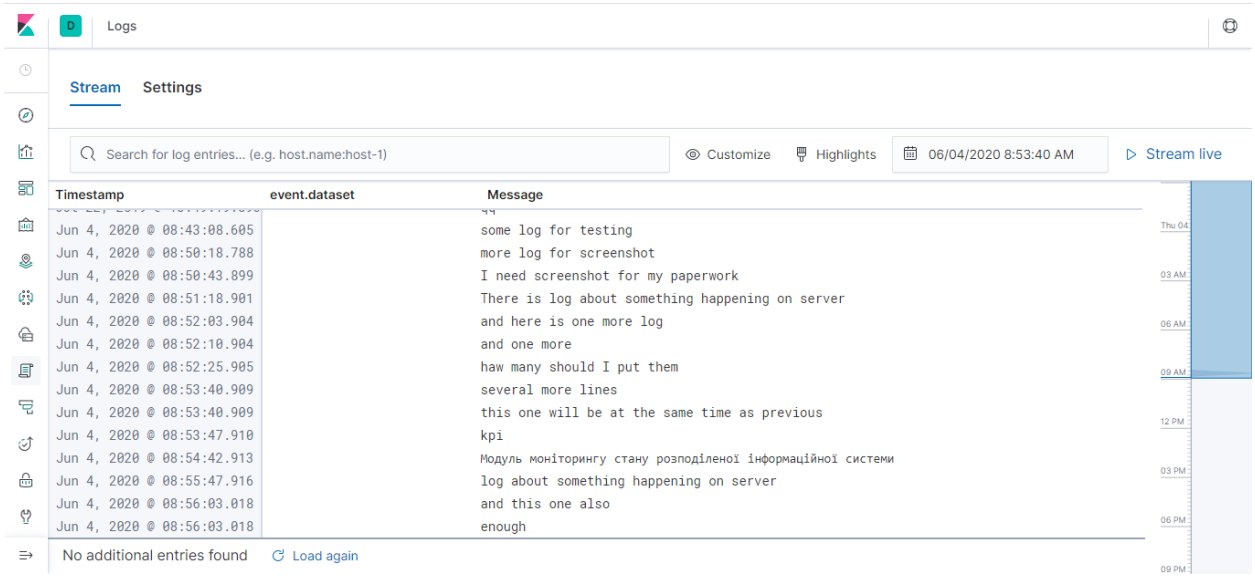


Рис. 3.4. Перегляд логів у налаштованому стеку ELK

ВИСНОВКИ ДО РОЗДІЛУ 3

Було розроблено модуль на платформі Node.js для моніторингу розподілених систем. Даний модуль дозволяє автоматизувати налаштування моніторингу у розподіленій системі.

Основними компонентами модуля є:

- два інтерфейси на вибір оператора;
- конфігуратор, який відповідає за процес налаштування;
- компонент, який відповідає за використання протоколу Telnet;
- сканер для сканування доступних вузлів;
- запис логів процесу налаштування.

Після отримання налаштувань від інтерфейсу, конфігуратор перевіряє його коректність та виконує за допомогою протоколу Telnet необхідні налаштування вузлів.

Результатом роботи модуля є налаштований стек ELK для моніторингу розподіленої системи. Результати моніторингу доступні для перегляду за допомогою інструменту Kibana.

ВИСНОВКИ

У даній роботі було розглянуто існуючі інструменти для моніторингу та виконано їх порівняння. Було визначено, що існує проблема автоматизації налаштування моніторингу у розподілених системах, а наявні способи автоматизації не завжди задовольнять потребам (автоматизація не повна або для автоматизації треба призупиняти роботу серверів).

Для рішення даної проблеми було обрано стек ELK та був написаний модуль, який дозволяє автоматизовано встановлювати даний стек у розподілених інформаційних системах не втручаючись у їх основну роботу. Тобто стек встановлюється під час роботи всієї системи без необхідності зупиняти сервери, на яких відбувається налаштування.

ЛІТЕРАТУРА

1. Rob Ewaschuk, Monitoring Distributed Systems // Site Reliability Engineering: How Google Runs Production Systems. - 2016. - Chapter 6.
2. Łukasz Kufel. Tools for distributed systems monitoring // Foundations of computing and decision sciences. - 2016. - No. 4, P.237-260.
3. ELK Stack: Elasticsearch, Logstash, Kibana [Електронний ресурс] // Elastic. - 2020. - Режим доступу до ресурсу: https://www.elastic.co/what-is/elk-stack.
4. Zabbix :: The Enterprise-Class Open Source Network Monitoring Solution [Електронний ресурс] // Zabbix. - 2020. - Режим доступу до ресурсу: https://www.zabbix.com.
5. Nagios - Network, Server and Log Monitoring Software [Електронний ресурс] // Nagios. - 2020. - Режим доступу до ресурсу: https://www.nagios.com.
6. Application Monitor | Application Monitoring Tools - ManageEngine Applications Manager [Електронний ресурс] // ManageEngine. - 2020. - Режим доступу до ресурсу: https://www.manageengine.com/products/applications_manager/
7. IBM SmartCloud Monitoring [Електронний ресурс] // IBM. - 2020. - Режим доступу до ресурсу: https://www.ibm.com/support/knowledgecenter/en/SS4EKN
8. IBM SmartCloud Monitoring 7.2 Solution Guide // IBM. - 2013. - 38 pages.
9. InfluxDB 1.X: Open Source Time Series Platform | InfluxData [Електронний ресурс] // InfluxData. - 2020. - Режим доступу до ресурсу: https://www.influxdata.com/time-series-platform.
10. Agent and data collector deployment [Електронний ресурс] // IBM. - 2020. - Режим доступу до ресурсу:

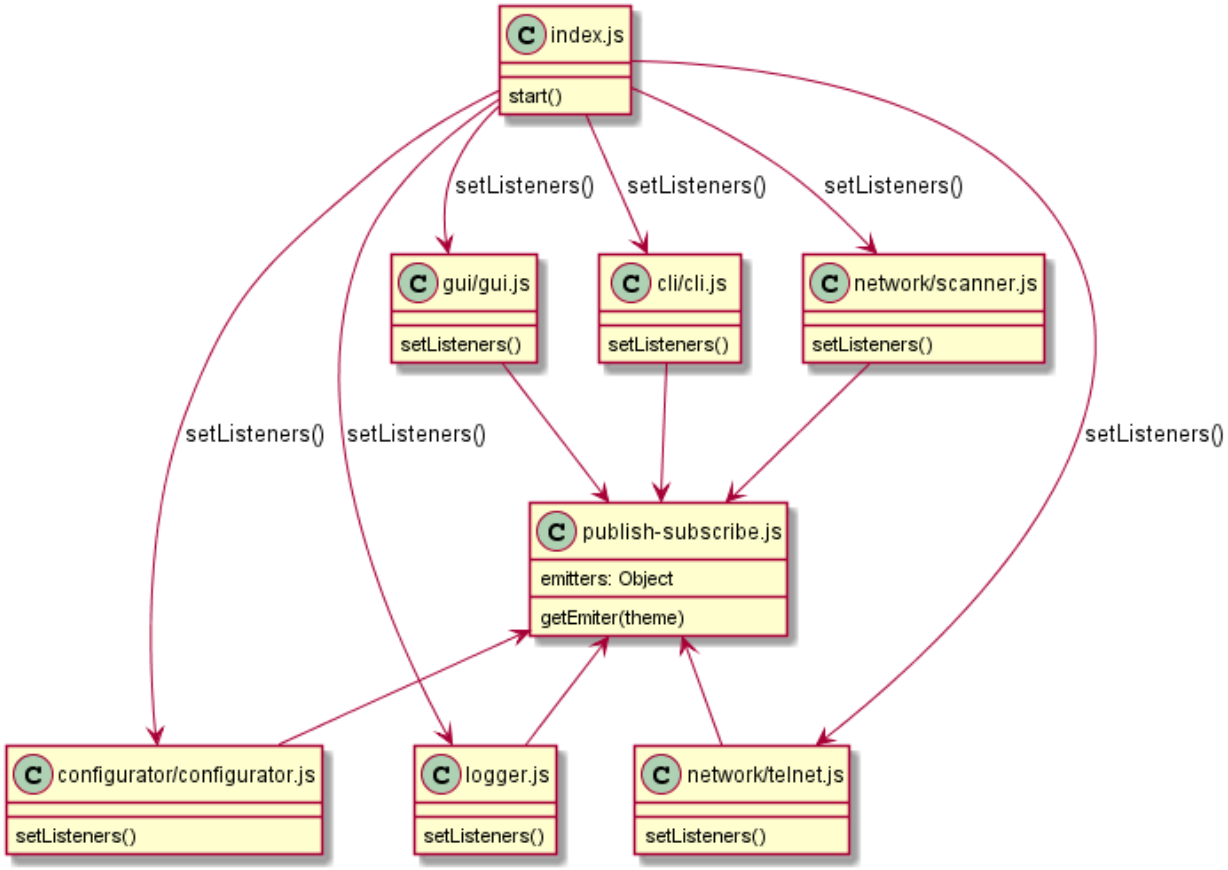
https://www.ibm.com/support/knowledgecenter/SSHLNR_8.1.4/com.ibm.pm.doc/install/planning_agents.html.

11. Mikell Groover. Fundamentals of Modern Manufacturing: Materials, Processes, and Systems. - 2014. - 929 pages.
12. Automating ELK Stack Installation [Електронний ресурс] // Lance Zukel. - 2020. - Режим доступу до ресурсу: https://lzukel.wordpress.com/2016/08/13/automating-elk-stack-installation.
13. Andrew Wheen. Dot-dash to Dot.Com: How Modern Telecommunications Evolved from the Telegraph to the Internet. Springer, 2011. - 320 pages.
14. Christoph Meinel; Harald Sack. Internetworking: Technological Foundations and Applications. X.media.publishing, 2013. - 916 pages.
15. The Story of the PING Program [Електронний ресурс] // Mike Muuss. - 2010. - Режим доступу до ресурсу: http://ftp.arl.army.mil/~mike/ping.html.
16. Wayne P. Stevens, Glenford J. Myers, Larry LeRoy Constantine. Structured design // IBM Systems Journal, 1974. - pages 115–139.
17. Edward Yourdon, Larry LeRoy Constantine. Structured Design: Fundamentals of a Discipline of Computer Program and Systems Design // Yourdon Press, 1979. - 446 pages.
18. Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides. Design Patterns: Elements of Reusable Object-Oriented Software // Addison Wesley, 1994.
19. Birman, K., Joseph, T. Exploiting virtual synchrony in distributed systems // Proceedings of the Eleventh ACM Symposium on Operating Systems Principles, 1987. - No. 87, pages 123-138.
20. Miško Hevery. Clean Code Talks - Global State and Singletons [Електронний ресурс] // Google Testing Blog. - 2008. - Режим

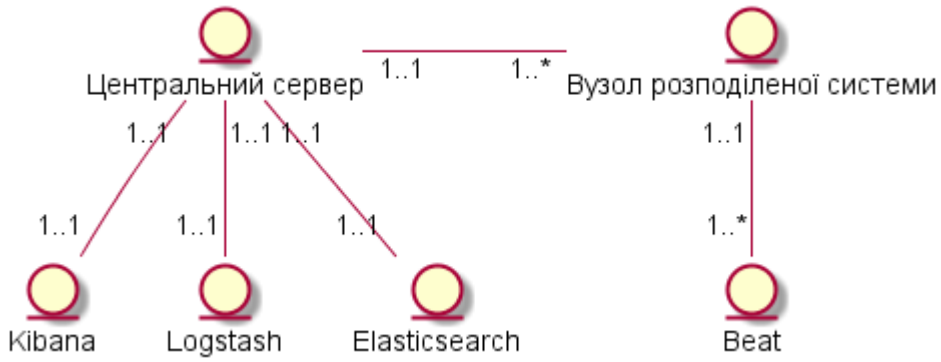
доступу до ресурсу: https://testing.googleblog.com/2008/11/clean-code-talks-global-state-and.html

21. Node.js [Електронний ресурс] // OpenJS Foundation. - 2020. - Режим доступу до ресурсу: https://nodejs.org/en.
22. Why Open Source Is Good For Business [Електронний ресурс] // Vlad V. - 2015. - Режим доступу до ресурсу: https://rubygarage.org/blog/why-open-source-is-good-for-business.
23. Six reasons why you might consider open source [Електронний ресурс] // Opensource.com. - 2015. - Режим доступу до ресурсу: https://opensource.com/life/15/12/why-open-source.
24. npm | build amazing things [Електронний ресурс] // npm. - 2020. - Режим доступу до ресурсу: https://www.npmjs.com.
25. Nmap: the Network Mapper - Free Security Scanner [Електронний ресурс] // Gordon Lyon. - 2020. - Режим доступу до ресурсу: https://nmap.org.
26. JSON Schema | The home of JSON Schema[Електронний ресурс] // JSON Schema. - 2020. - Режим доступу до ресурсу: https://json-schema.org.
27. JSON Schema Based Editor[Електронний ресурс] // Github. - 2020. - Режим доступу до ресурсу: https://github.com/json-editor/json-editor.
28. Empowering App Development for Developers | Docker[Електронний ресурс] // Docker. - 2020. - Режим доступу до ресурсу: https://www.docker.com.

					ІАЛЦ.466514.004 Д1	
Зм.	Арк.	№ докум.	Підпис	Дата		
Розробив		Лисенко Д. В.			Модуль моніторингу стану розподіленої інформаційної системи	Літ. / Аркуш 1 / Аркушів 1
Перевірив		Болдак А. О.				
Реценз.					Алгоритм роботи модуля	НТУУ КПІ ім. Ігоря Сікорського, ФІОТ, ІО-61
Н. Контр.		Сімоненко В. П.				
Затвердив						



					ІАЛЦ.466514.005 Д2		
Зм.	Арк.	№ докум.	Підпис	Дата			
Розробив		Лисенко Д. В.			Модуль моніторингу стану розподіленої інформаційної системи Діаграма класів модуля	Літ.	Аркуш 1 / Аркушів 1
Перевірив		Болдак А. О.					
Реценз.						НТУУ КПІ ім. Ігоря Сікорського, ФІОТ, ІО-61	
Н. Контр.		Сімоненко В. П.					
Затвердив							



					ІАЛЦ.466514.006 ДЗ		
Зм.	Арк.	№ докум.	Підпис	Дата			
Розробив		Лисенко Д. В.			Модуль моніторингу стану розподіленої інформаційної системи	Літ.	Аркуш 1 / Аркушів 1
Перевірив		Болдак А. О.					
Реценз.					Діаграма сутностей	НТУУ КПІ ім. Ігоря Сікорського, ФІОТ, ІО-61	
Н. Контр.		Сімоненко В. П.					
Затвердив							